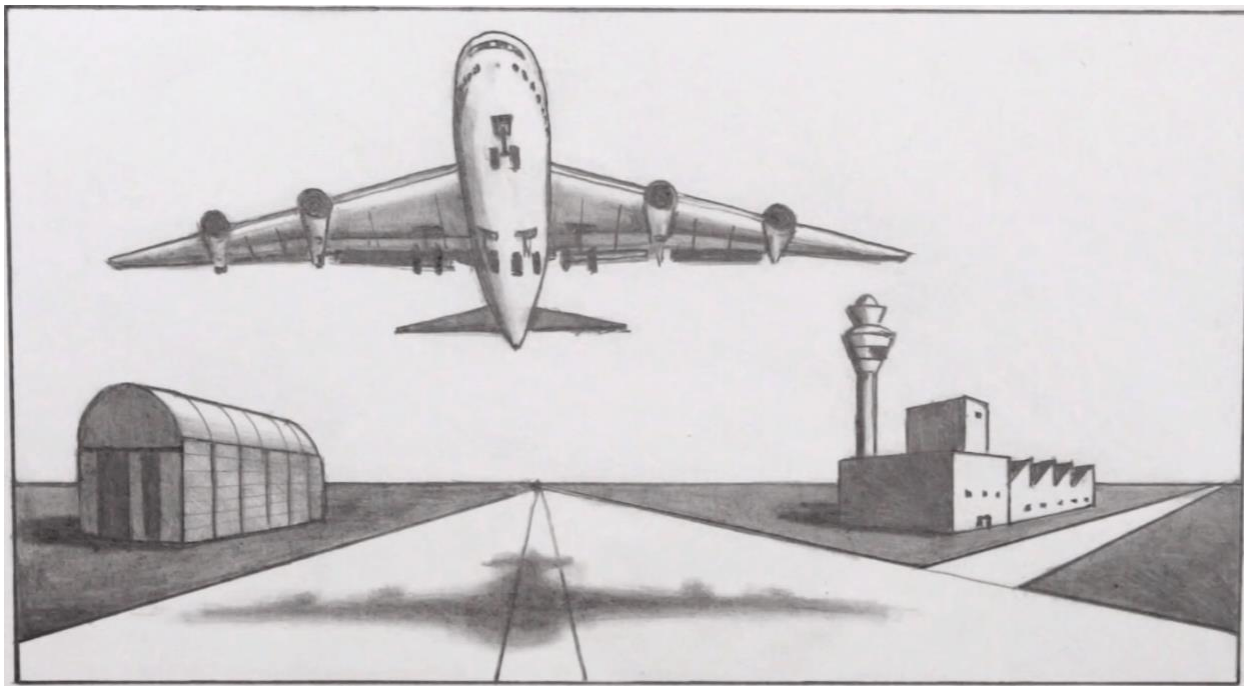
Итоговая работа
по курсу «SQL и получение данных»

Работа выполнена в качестве итогового задания по итогам прохождения курса обучения «SQL и получение данных».

В работе использована база данных PostgreSQL по бронированиям авиабилетов на рейсы авиакомпании.

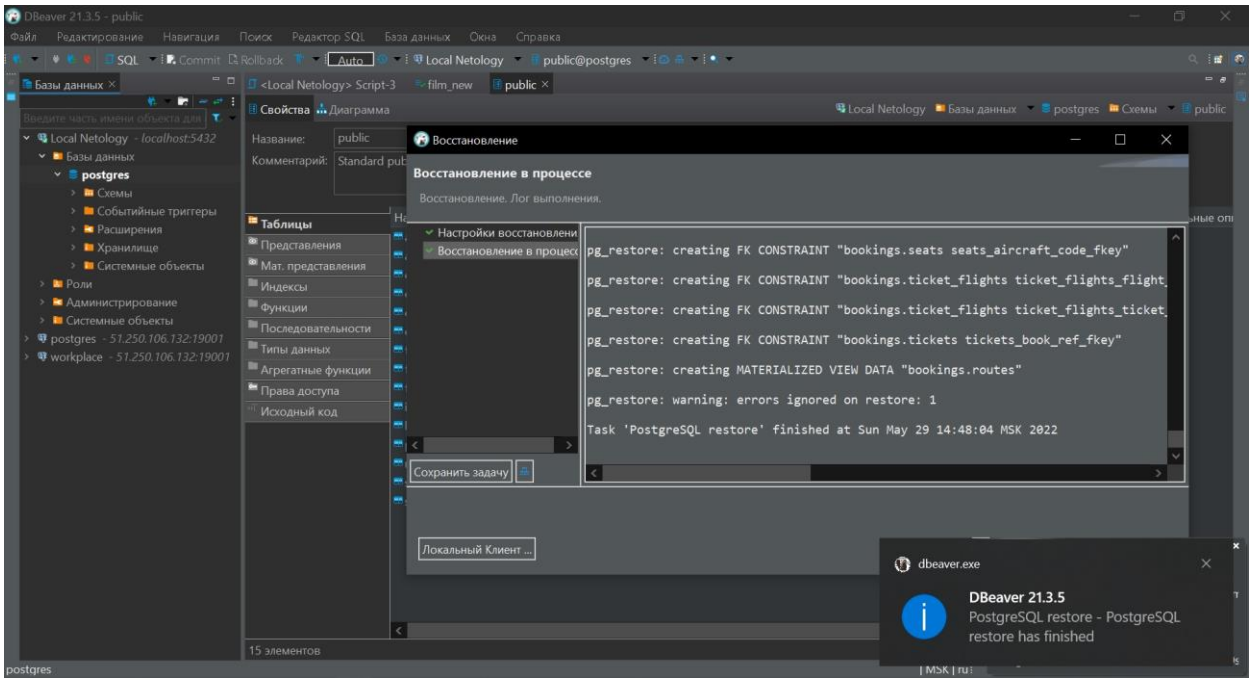
Работа с базой осуществлялась с использованием SQL-клиента DBeaver (v. 21.3.5).



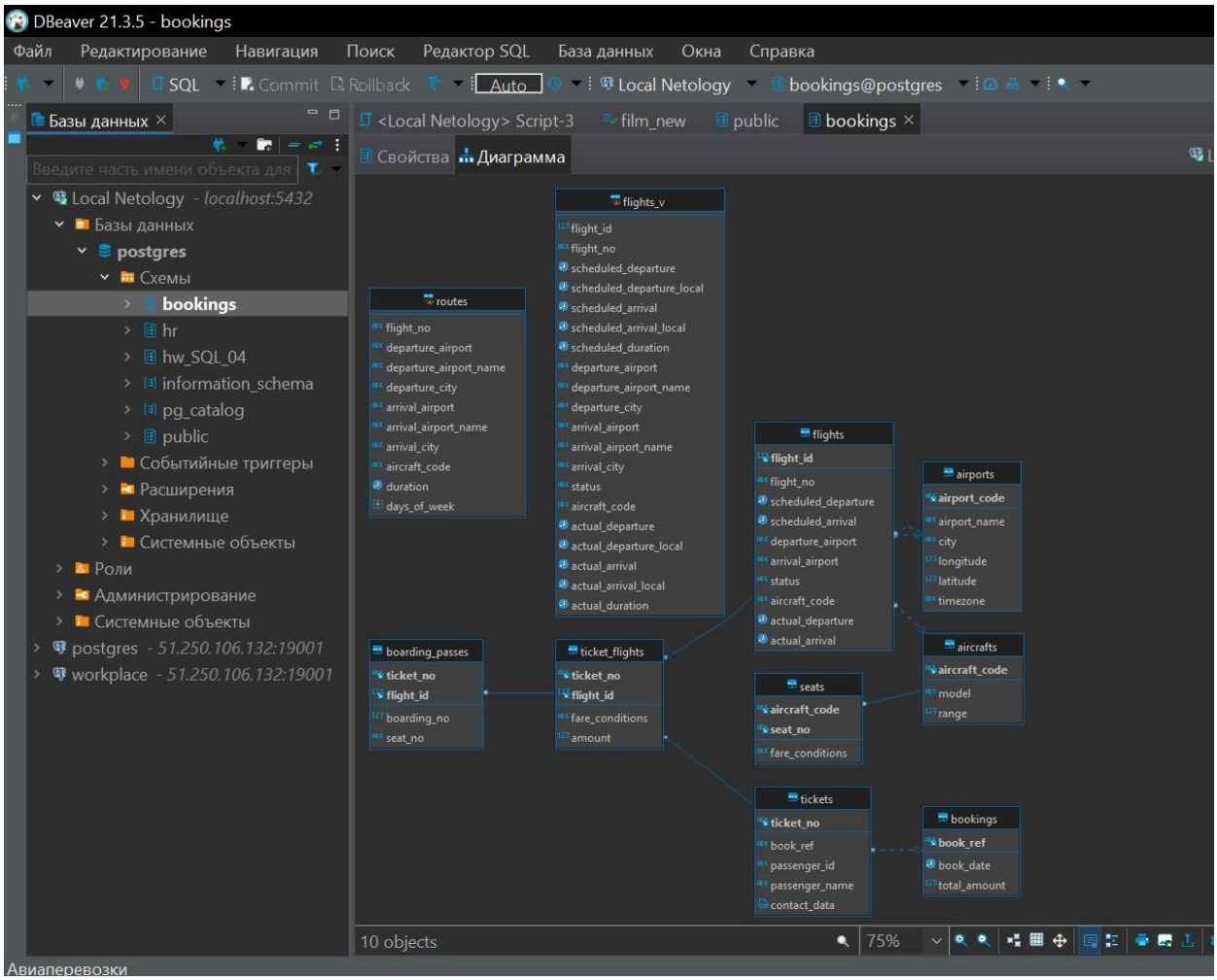
Испытай один раз полет, и твои глаза навечно будут устремлены в небо. Однажды там побывав, на всю жизнь ты обречен тосковать о нем (с) Леонардо Да Винчи

1. В работе использовался локальный тип подключения (восстановление БД из .backup-файла).

Скриншот успешного восстановления БД:



2. Скриншот ER-диаграммы из DBeaver:



3. Краткое описание БД - из каких таблиц и представлений состоит.

База данных (БД) состоит из 8 таблиц (отношений) и 2 представлений.

Перечень таблиц (отношений) БД:
- airports *(аэропорты)*
- aircrafts *(самолеты)*
- flights *(рейсы)*
- seats *(посадочные места)*
- bookings *(бронирования)*
- tickets *(билеты)*
- ticket_flights *(перелеты – таблица отношений)*
- boarding passes *(посадочные талоны)*

Перечень представлений БД:
- routes *(информация о маршрутах)*
- flights_v *(информация о рейсах)*

4. Развернутый анализ БД (описание таблиц, логики, связей и бизнес области). Бизнес-задачи, которые можно решить, используя БД.

Основная сущность БД - бронирование (bookings).

Отношение “bookings”:
- первичный ключ – book_ref
- содержит данные о бронированиях авиабилетов
- каждое бронирование имеет уникальный идентификатор
- одно бронирование может содержать несколько пассажиров (билетов)
- book_ref является внешним ключом для поля book_ref отношения tickets
- поле total_amount хранит общую стоимость включенных в бронирование перелетов всех пассажиров.

Отношение “tickets”:
- первичный ключ – ticket_no
- ограничение внешнего ключа по полю book_ref (родительская таблица – bookings)
- содержит информацию о приобретенных билетах
- каждый билет имеет уникальный идентификатор
- несколько билетов могут относиться к одному и тому же бронированию в таблице bookings
- один билет может содержать как один, так и несколько перелетов (таблица flights)

- билет содержит идентификатор пассажира (passenger_id) — номер документа, удостоверяющего личность, — его фамилию и имя (passenger_name) и контактную информацию (contact_date)

- ни идентификатор пассажира, ни имя не являются постоянными (можно поменять паспорт, можно сменить фамилию), поэтому однозначно найти все билеты одного и того же пассажира невозможно

Отношение “airports”:

- первичный ключ – airport_code

- содержит справочник аэропортов

- каждому аэропорту соответствует уникальный идентификатор

- для города не предусмотрено отдельной сущности, но название (city) указывается и может служить для того, чтобы определить аэропорты одного города

- также указывается широта (longitude), долгота (latitude) и часовой пояс (timezone)

Отношение “aircrafts”:

- первичный ключ - aircraft_code

- содержит справочник самолетов

- каждая модель самолета имеет различную схему рассадки (таблица seats)

- каждая модель воздушного судна идентифицируется своим трехзначным кодом (aircraft_code)

- указывается также название модели (model) и максимальная дальность полета в километрах (range)

Отношение “seats”:

- первичный ключ – составной - aircraft_id, seat_no

- ограничение внешнего ключа по полю aircraft_code (родительская таблица - aircrafts)

- содержит информацию о количестве мест в каждой из моделей самолета

- каждое место имеет закрепленный за ним класс обслуживания (fare_conditions) — Economy, Comfort или Business

Отношение “flights”:

- первичный ключ – flight_id

- ограничения внешнего ключа по полям departure_airport и arrival_airport (аэропорты вылета и прибытия, родительская таблица – airports (airport_code))

- ограничение внешнего ключа по полю aircraft code (родительская таблица – aircrafts (aircraft_code))

- содержит информацию о рейсах

- каждый рейс (flights) следует из одного аэропорта (airports) в другой

- рейсы с одним номером имеют одинаковые пункты вылета и назначения, но будут отличаться датой отправления

- понятие «рейс с пересадками» отсутствует: если из одного аэропорта до другого нет прямого рейса, в билет включаются несколько необходимых рейсов

- у каждого рейса есть запланированные дата и время вылета (scheduled_departure) и прибытия (scheduled_arrival)

- реальные время вылета (actual_departure) и прибытия (actual_arrival) могут отличаться

- статус рейса (status) может принимать одно из следующих значений: *Scheduled / Рейс доступен для бронирования. Это происходит за месяц до плановой даты вылета; до этого запись о рейсе не существует в базе данных. • On Time Рейс доступен для регистрации (за сутки до плановой даты вылета) и не задержан. • Delayed Рейс доступен для регистрации (за сутки до плановой даты вылета), но задержан. • Departed Самолет уже вылетел и находится в воздухе. 6 • Arrived Самолет прибыл в пункт назначения. • Cancelled Рейс отменен*

Отношение “ticket-flights”:

- таблица отношений таблиц tickets и flights
- первичный ключ – составной – ticket_no, flight_id
- ограничения внешнего ключа по полям ticket_no (родительская таблица tickets), flight_id (родительская таблица - flights)
- содержит информацию о перелетах
- 1 билет в таблице tickets может содержать 1 или несколько перелетов в таблице ticket_flights
- для каждой записи указываются стоимость (amount) и класс обслуживания (fare_conditions)

Отношение “boarding passes”:

- первичный ключ – составной – ticket_no, flight_id
- ограничения внешнего ключа по полям ticket_no и flight_id (родительская таблица – ticket_flights)
- содержит информацию о посадочных талонах
- пассажир может зарегистрироваться только на тот рейс, который есть у него в билете
- посадочным талонам присваиваются последовательные номера (boarding_no) в порядке регистрации пассажиров на рейс (этот номер будет уникальным только в пределах данного рейса)
- в посадочном талоне указывается номер места (seat_no)

Представление “flights_v” содержит информацию:

- расшифровку данных об аэропорте вылета (departure_airport, departure_airport_name, departure_city)
- расшифровку данных об аэропорте прибытия (arrival_airport, arrival_airport_name, arrival_city)
- местное время вылета (scheduled_departure_local, actual_departure_local)
- местное время прибытия (scheduled_arrival_local, actual_arrival_local)
- продолжительность полета (scheduled_duration, actual_duration).

Материализованное представление “routes” содержит информацию:

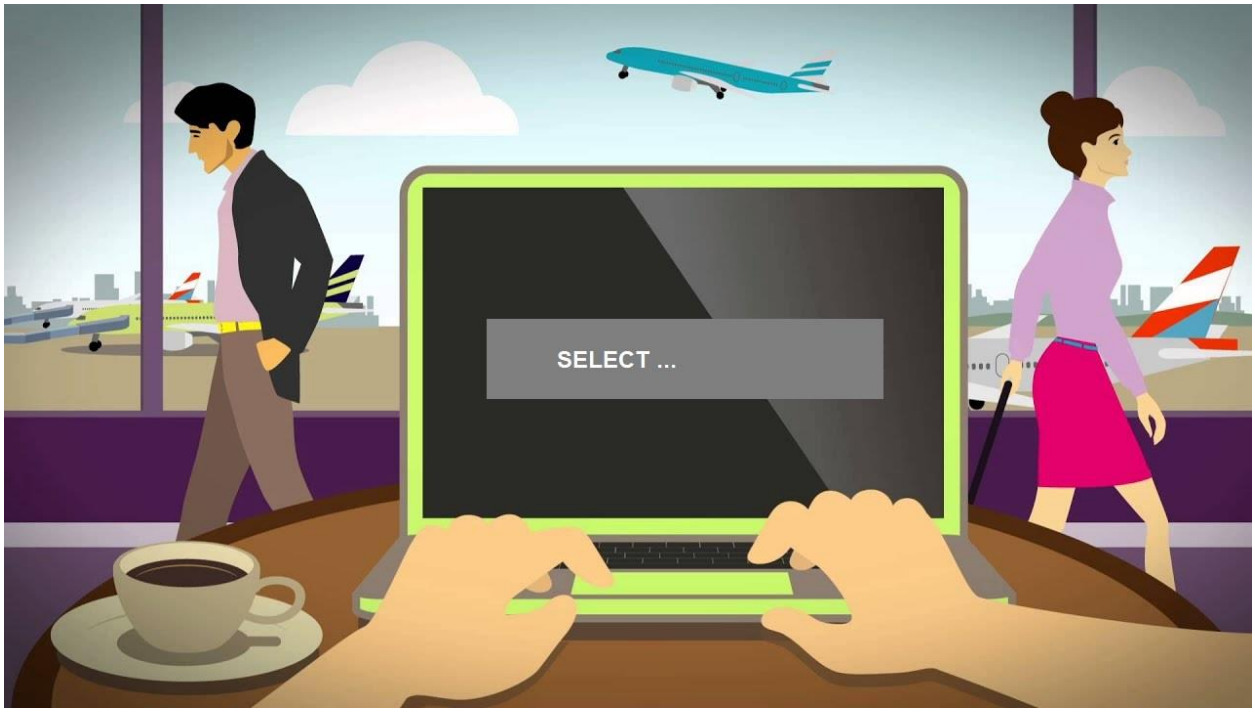
- информация из таблицы рейсов о маршруте (номер рейса, аэропорты отправления и назначения), которая не зависит от конкретных дат рейсов.

Бизнес-область Базы данных – деятельность авиакомпании по организации перевозки авиапассажиров.

БД позволяет решать задачи:

- контроль за составом авиапарка;
- поиск подходящих авиарейсов;
- планирование оптимальных маршрутов между аэропортами;
- ведение процесса продажи билетов;
- ведение процесса выдачи посадочных талонов;
- различные виды аналитики (популярность маршрутов, заполняемость самолетов по рейсам, стоимость перелетов, типичный портрет пассажира на каждом из направлений, заполняемость самолетов по дням недели / месяцам, самые дорогие / дешевые бронирования, и т.п.);
- а также прочие задачи, связанные с бизнес-областью БД.

5. Список SQL-запросов с описанием логики их выполнения.



-- Запрос № 1. В каких городах больше одного аэропорта? --

```
select city as "Город", count(airport_code) as "Количество аэропортов"
from airports
group by city
having count(airport_code) > 1
order by 2 desc
```

Описание логики:

Из таблицы «Airports» выводим атрибуты «city» (город) и количество аэропортов в каждом городе (с помощью функции «count»).

Группируем выдачу по названию города.

Фильтруем результаты запроса по количеству аэропортов (в рамках оператора «having»), выбирая только те города, где более 1 аэропорта.

Сортируем выдачу по количеству аэропортов в порядке убывания (в рамках оператора «order by»).

-- Запрос № 2. В каких аэропортах есть рейсы, выполняемые самолетом с максимальной дальностью перелета? (Подзапрос) --

```
select distinct airport_name as "Название аэропорта", airport_code as "Код аэропорта", city as "Город"
from airports a
join flights f on a.airport_code = f.departure_airport
join aircrafts a2 on a2.aircraft_code = f.aircraft_code
```

```
where a2.aircraft_code = (select a3.aircraft_code from aircrafts a3 order by range desc limit 1)
order by 3, 1
```

Описание логики:

Таблицу «Airports» обогащаем данными из таблиц «Flight» и «Aircrafts».

Оператором «Where» фильтруем выдачу с помощью подзапроса для выделения самолета с наибольшей дальностью перелета («range»).

Выводим атрибуты – название аэропорта, код аэропорта, город. Сортируем выдачу по городу и названию аэропорта.

-- Запрос № 3. Вывести 10 рейсов с максимальным временем задержки вылета (Оператор LIMIT) --

```
select flight_id, flight_no, departure_airport, arrival_airport, (actual_departure -
scheduled_departure) as "Departure delay"
from flights
where status in ('Departed', 'Arrived')
order by 5 desc
limit 10
```

Описание логики:

Работаем с таблицей «Flights». Исключаем из выборки рейсы, которые еще не улетели (оставляем status либо «Departed», либо «Arrived»).

Выводим идентификатор рейса, номер рейса, аэропорт отправления, аэропорт прибытия, разницу между фактическим временем вылета («actual_departure») и плановым временем вылета («scheduled_departure») – т.е. задержку вылета.

Сортируем по задержке вылета (по убыванию). Оставляем в выдаче 10 рейсов с наибольшим значением задержки вылета.

-- Запрос № 4. Были ли брони, по которым не были получены посадочные талоны? (Верный тип JOIN) --

```
select distinct t.book_ref
from bookings b
left join tickets t on b.book_ref = t.book_ref
left join boarding_passes bp on t.ticket_no = bp.ticket_no
where bp.boarding_no is null
order by 1
```

Описание логики:

За базовую таблицу берем «bookings». Чтобы выявить брони, по которым не были получены посадочные талоны, обогащаем указанную таблицу через «left join» данными из таблиц «tickets» и «boarding_passes».

Проводим фильтрацию - оставляем в выдаче только те бронирования, по которым не получены посадочные талоны.

Группируем по идентификационному номеру бронирования. Выводим уникальные идентификаторы бронирований, удовлетворяющих условию.

-- Запрос № 5. Найдите количество свободных мест для каждого рейса, их % отношение к общему количеству мест в самолете.

-- Добавьте столбец с накопительным итогом - суммарное накопление количества вывезенных пассажиров из каждого аэропорта на каждый день.

-- Т.е. в этом столбце должна отражаться накопительная сумма - сколько человек уже вылетело из данного аэропорта на этом или более ранних рейсах в течении дня

-- (Оконная функция, Подзапросы или/и cte) --

```
with cte_ts as
   (
   select flight_id, count(seat_no) as taken_seats
   from boarding_passes bp
   group by flight_id
   ),
 cte_os as
   (
   select aircraft_code, count(s.seat_no) as overall_seats
   from seats s
   group by aircraft_code
   )
select f.flight_id as "id перелета", f.flight_no as "Номер рейса", departure_airport as
"Аэропорт вылета",
        actual_departure as "Время вылета (факт)", overall_seats as "Всего мест", overall_seats
- taken_seats as "Свободно мест",
        (round (((overall_seats - taken_seats)::numeric / overall_seats::numeric), 2) * 100) as "%
свободных мест",
        sum (taken_seats) over (partition by departure_airport, date_trunc('day', actual_departure)
order by actual_departure) as "К-во вывез. пасс-ров (накоп. ежедн.)"
from cte_ts
join flights f on cte_ts.flight_id = f.flight_id
join cte_os on cte_os.aircraft_code = f.aircraft_code
join airports a2 on f.departure_airport = a2.airport_code
order by departure_airport, actual_departure asc
```

Описание логики:

Создаем 2 обобщенных табличных выражения (CTE): первое - на основе таблицы «boarding_passes» с данными по занятым местам на каждом рейсе, второе – на основе таблицы «seats» с данными по общему количеству мест в каждом самолете.

Далее в рамках основной части запроса обогащаем первое CTE данными из таблицы «flights», присоединяем второе CTE, обогащаем данными из таблицы airports.

Выводим данные по идентификационному номеру перелета, номеру рейса, аэропоту вылета, фактическому времени вылета, общему количеству мест в самолете, количеству свободных мест, % свободных мест (считаем с промежуточным приведением данных к типу «numeric» с округлением до 2 знаков после запятой), далее – используем оконную функцию для вывода накопительного количества вывезенных пассажиров из каждого из аэропортов в каждый из дней (для этого используем функцию SUM по атрибуту «taken seats», используем оператор «partition by» по полям с аэропортом вылета и округленным до дня временем вылета с сортировкой по времени вылета).

Получившийся результат сортируем по аэропорту вылета, времени вылета (в нарастающем порядке).

-- Запрос № 6. Найдите процентное соотношение перелетов по типам самолетов от общего количества (Подзапрос или окно, Оператор ROUND) --

```
select distinct on (a.model) a.model as "Тип самолета",
        round((count( f.flight_id) over(partition by a.model)::numeric / count(f.flight_id) over()::numeric),2) * 100 as "Доля перелетов, %"
from aircrafts a
left join flights f using (aircraft_code)
group by 1, f.flight_id
order by 1
```

Описание логики:

Таблицу «Aircrafts» обогащаем данными из таблицы «Flights» (через left join, чтобы не потерять данные по самолетам, которые не совершали рейсов, если такие есть).

Выводим типы самолетов (только уникальные – с помощью оператора «distinct») и долю перелетов (в %), приходящуюся на каждый из типов.

Для расчета доли перелетов используем оконную функцию с функцией «count» (для расчета количества перелетов) и выводом данным по каждому типу самолета (с помощью «partition by»). Для корректности расчета приводим результат к типу «numeric», делим на общее количество перелетов для всех типов самолетов через «окно» без «partition by» (также приводим к типу «numeric»), округляем до 2 знаков после запятой и умножаем на 100 (для получения размерности результата в процентах).

Группируем результат запроса по типам самолетов, идентификатору перелета. Сортируем по типам самолетов.

-- Запрос № 7. Были ли города, в которые можно добраться бизнес - классом дешевле, чем эконом-классом в рамках перелета? (CTE) --

```
with cte_1 as (
        select tf.flight_id, tf.fare_conditions, tf.amount as Amount_Business
        from ticket_flights tf
```

```
        where tf.fare_conditions = 'Business'
        ),
    cte_2 as (
        select tf2.flight_id, tf2.fare_conditions, tf2.amount as Amount_Economy
        from ticket_flights tf2
        where tf2.fare_conditions = 'Economy'
        )
select city as "Город", flight_no as "№ рейса (перелета)", Amount_Economy as "Стоимость эконом-класса", Amount_Business as "Стоимость бизнес-класса"
from flights f
join cte_1 on f.flight_id = cte_1.flight_id
join cte_2 on f.flight_id = cte_2.flight_id
join airports a on a.airport_code = f.arrival_airport
where Amount_Economy > Amount_Business
group by 1, 2, 3, 4
```

Описание логики:

Создаем 2 обобщенных табличных выражения (CTE) на основе таблицы «ticket-flights»: первое – только для билетов бизнес-класса, второе – только для билетов эконом-класса. В выдаче указанных CTE указываем идентификатор рейса, условия перелета (эконом или бизнес), стоимость билетов.

Далее в рамках основной части запроса обогащаем таблицу «flights» данными из двух CTE, производя объединение таблиц по полю «flight_id», и из таблицы «airports» (т.к. в условии указано «города, в которые можно добраться…», то объединяем по атрибуту «arrival_airport»).

Отфильтровываем результат по записям, где стоимость билетов эконом-класса больше стоимости билетов бизнес-класса.

Выводим в результатах запроса город, номер рейса, стоимость билетов эконом-класса, стоимость билетов бизнес-класса.

Группируем по всем указанным выше полям.

По итогам выполнения запроса – города, в которые можно добраться бизнес-классом дешевле, чем эконом-классом, отсутствуют.

-- Запрос № 8. Между какими городами нет прямых рейсов? (Декартово произведение в предложении FROM, самостоятельно созданные представления (если облачное подключение, то без представления), Оператор EXCEPT) --

```
create view cities_combination as
  (select a.city as dep_city, a2.city as arr_city
  from airports a, airports a2
  where a.city > a2.city)

select *
from cities_combination
```

```
except
select departure_city, arrival_city
from routes
order by 1, 2
```

Описание логики:

Создаем представление с полным перечнем комбинаций городов вылета и прилета (декартово произведение с исключением «зеркальных» пар городов).

В основном запросе с помощью оператора «except» удаляем из этой таблицы все записи, соответствующие реально существующим сочетаниям города вылета и города прилета из существующего в БД представления «routes».

В результате в выдаче остаются только пары городов, не связанные прямыми рейсами.

Сортируем результат по городу вылета и городу прибытия.

-- Запрос № 9. Вычислите расстояние между аэропортами, связанными прямыми рейсами, сравните с допустимой максимальной дальностью перелетов в самолетах, обслуживающих эти рейсы * (Оператор RADIANS или sind/cosd, CASE) --

```
with cte_departure_coordinates as
 (
 select flight_id, departure_airport, longitude as departure_longitude, latitude as
departure_latitude
 from flights f
 join airports a on a.airport_code = f.departure_airport
 ),
   cte_arrival_coordinates as
 (
 select flight_id, arrival_airport, longitude as arrival_longitude, latitude as arrival_latitude
 from flights f2
 join airports a2 on a2.airport_code = f2.arrival_airport
 ),
   cte_combined as
 (
 select f3.flight_id, aircraft_code, f3.departure_airport, departure_longitude, departure_latitude,
f3.arrival_airport, arrival_longitude, arrival_latitude,
                        round((6371*(acos(sind(departure_latitude)*sind(arrival_latitude) +
cosd(departure_latitude)*cosd(arrival_latitude)*cosd(departure_longitude -
arrival_longitude))))::numeric,0) as distance_km
 from flights f3
 join cte_departure_coordinates on cte_departure_coordinates.flight_id = f3.flight_id
 join cte_arrival_coordinates on cte_arrival_coordinates.flight_id = f3.flight_id
 where f3.departure_airport = cte_departure_coordinates.departure_airport and f3.arrival_airport
= cte_arrival_coordinates.arrival_airport
 )
```

```
select distinct on (departure_airport, arrival_airport) departure_airport, arrival_airport,
  distance_km,
  range,
  case when distance_km <= a3.range then 'Enough range'
  else 'Not enough range'
  end as range_sufficiency
from cte_combined
join aircrafts a3 on a3.aircraft_code = cte_combined.aircraft_code
order by 1,2
```

Описание логики:

Создаем 3 обобщенных табличных выражения (CTE):
- первое – на базе таблицы «flights» по аэропортам вылета с обогащением координатами из таблицы «airports»;
- второе – аналогичное по аэропортам прибытия;
- третье – совмещенное из таблицы «flight», первых двух CTE, в выдаче которого имеются аэропорты вылета и прибытия и их координаты, атрибуты «flight_id» и «aircraft_code» (понадобится для последующего подтягивания атрибута «range»), а также расстояние между аэропортами в км (рассчитано по формуле, приведенной в формулировке задания, с округлением до целых чисел).
В основной части запроса к 3-ему CTE присоединяем таблицу «aircraft_id».
Выводим:
- уникальные наборы пары значений аэропорта вылета и аэропорта прибытия;
- расстояние между ними;
- максимальную дальность перелета самолетов, осуществляющих перелеты между приведенными аэропортами;
- результат сравнения расстояния между аэропортами и максимальной дальности перелета самолетов (с использованием оператора CASE).
Результат сортируется по названиям аэропортов вылета и прибытия.