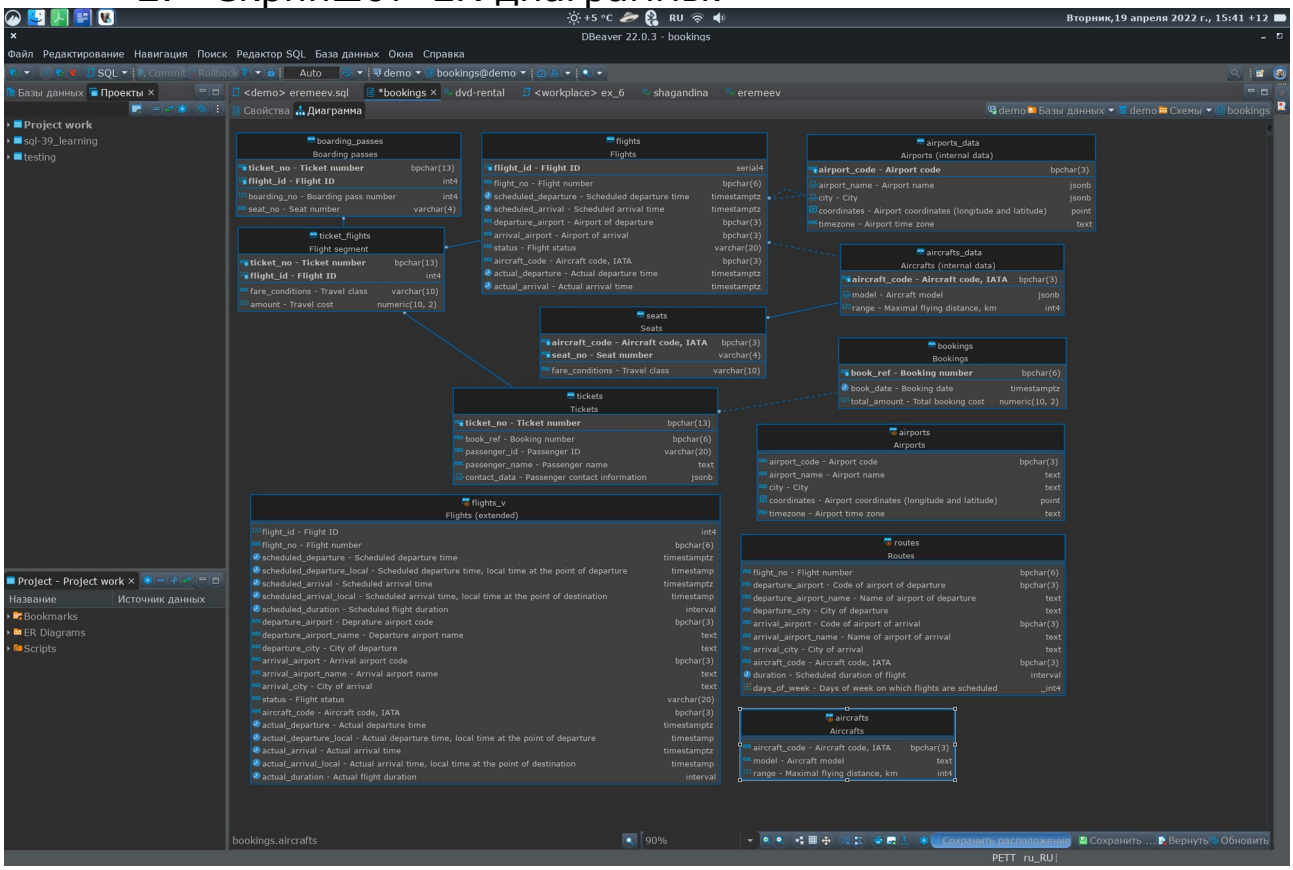
Еремеев С.И.

Проектная работа по модулю «SQL и получение данных»

1. В работе использовалось облачное подключение.
2. Скриншот ER-диаграммы.



3. Для выполнения работы использовалась [демонстрационная база данных] для СУБД PostgreSQL, представленная компанией [Postgres Professional]. В качестве предметной области выбраны авиаперевозки по России.

3.1.Краткое описание базы данных

[Демонстрационная база данных] состоит из восьми таблиц, трёх представлений и одного материализованного представления.

Таблицы:

aircrafts — самолёты;
airports — аэропорты;
boarding_passes — посадочные талоны;
bookings — бронирования;
flights — рейсы;
seats — места;

ticket_flights — перелёты;
tickets — билеты.
Материализованное представление:
routes — маршруты.
Представления:
flights_v — рейсы;
airports — аэропорты;
aircrafts — самолёты.

4. Развёрнутый анализ базы данных

Основной сущностью является бронирование (*bookings*). В одно бронирование можно включить несколько пассажиров, каждому из которых выписывается отдельный билет (*tickets*). Билет имеет уникальный номер и содержит информацию о пассажире. Как таковой пассажир не является отдельной сущностью. Как имя, так и номер документа пассажира могут меняться с течением времени, так что невозможно однозначно найти все билеты одного человека; для простоты можно считать, что все пассажиры уникальны.

Билет включает один или несколько перелётов или рейсов (*ticket_flights*), образуя для пассажира единый перевозочный документ. Несколько перелётов могут включаться в билет в случаях, когда нет прямого рейса, соединяющего пункты отправления и назначения (полет с пересадками), либо когда билет взят «туда и обратно». В схеме данных нет жёсткого ограничения, но предполагается, что все билеты в одном бронировании имеют одинаковый набор перелётов.

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.

При регистрации на рейс пассажиру выдаётся посадочный талон (*boarding_passes*), в котором указано место в самолёте. Пассажир может зарегистрироваться только на тот рейс, который есть у него в билете. Комбинация рейса и места в самолёте должна быть уникальной, чтобы не допустить выдачу двух посадочных талонов на одно место.

Количество мест (*seats*) в самолёте и их распределение по классам обслуживания зависит от модели самолёта (*aircrafts*), выполняющего рейс. Предполагается, что каждая модель самолёта имеет только одну компоновку салона. Схема

данных не контролирует, что места в посадочных талонах соответствуют имеющимся в самолёте.

4.1.Описание таблиц.

4.1.1. Таблица *bookings.aircrafts_data* (самолёты).

Каждая модель воздушного судна идентифицируется своим трёхзначным кодом (*aircraft_code*, ключевое поле, является внешним ключом для таблиц *flights* и *seats*). Указывается также название модели (*model*) и максимальная дальность полёта в километрах (*range*).

4.1.2. Таблица *bookings.airports_data* (аэропорты).

Аэропорт идентифицируется трёхбуквенным кодом (*airport_code*, ключевое поле, на которое также ссылаются поля *flights.arrival_airport* и *flights.departure_airport*) и имеет своё имя (*airport_name*). Для города не предусмотрено отдельной сущности, но название (*city*) указывается и может служить для того, чтобы определить аэропорты одного города. Также указываются географические координаты аэропорта — в последовательности долгота, широта (*coordinates*) и часовой пояс (*timezone*).

4.1.3. Таблица *bookings.boarding_passes* (посадочные талоны).

При регистрации на рейс, которая возможна за сутки до плановой даты отправления, пассажиру выдаётся посадочный талон. Он идентифицируется также, как и перелёт — номером билета и номером рейса. 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*). Таблица имеет составной ключ по полям *boarding_passes.ticket_no*, *boarding_passes.flight_id* и ограничения внешнего ключа по этим же полям, на которые ссылаются поля *ticket_flights.ticket_no* и *ticket_flights.flight_id*.

4.1.4. Таблица *bookings.bookings* (бронирования).

Пассажир заранее (*book_date*, максимум за месяц до рейса) бронирует билет себе и, возможно, нескольким другим пассажирам. Бронирование идентифицируется номером (*book_ref*, шестизначная комбинация букв и цифр, ключевое поле и ограничение внешнего ключа для поля *ticket.book_ref*). Поле *total_amount* хранит общую стоимость включённых в бронирование перелётов всех пассажиров.

4.1.5. Таблица *bookings.flights* (рейсы).

Естественный ключ таблицы рейсов состоит из двух полей — номера рейса (*flight_no*) и даты отправления (*scheduled_departure*). Чтобы сделать внешние ключи на эту таблицу компактнее, в качестве первичного используется суррогатный ключ (*flight_id*). Рейс всегда соединяет две точки — аэропорты вылета (*departure_airport*) и прибытия (*arrival_airport*). Такое понятие, как «рейс с пересадками» отсутствует: если из одного аэропорта до другого нет прямого рейса, в билет просто включаются несколько необходимых рейсов. У каждого рейса есть запланированные дата и время вылета (*scheduled_departure*) и прибытия (s*cheduled_arrival*). Реальные время вылета (*actual_departure*) и прибытия (*actual_arrival*) могут отличаться: обычно незначительно, но иногда и на несколько часов, если рейс задержан. Ограничения внешнего ключа: поле *flighs.aircraft_code* является внешним ключом для *aircrafts.aircraft_code*, f*light.arrival_airport* для *airports.airport_code*, *flight.departure_airport* для *airports.airport_code*; поле *flights.flight_id* ссылается как на внешний ключ на поле *ticket_flights.flight_id*.

4.1.6. Таблица *bookings.seats* (места).

Места определяют схему салона каждой модели. Каждое место определяется своим номером (*seat_no*) и имеет закрепленный за ним класс обслуживания (*fare_conditions*) — Economy, Comfort или Business. Составной естественный ключ таблицы: поля *seats.aircraft_code*, *seats.seat_no*; ограничение внешнего ключа для *aircrafts.aircraft_code* — поле *seats.aircraft_code*.

4.1.7. Таблица b*ookings.ticket_flights* (перелёты).

Перелёт соединяет билет с рейсом и идентифицируется их номерами. Для каждого перелёта указываются его стоимость (*amount*) и класс обслуживания. Перелёт можно определить как перевозку с позиции пассажира, тогда как рейс (предыдущая таблица) та же перевозка, но рассматриваемая с позиции перевозчика. Составной внешний ключ *ticket_flights.ticket_no, ticket_flights.flight_id*. Ограничения внешнего ключа:на *ticket_flights.flight_id* ссылается *flights.flight_id*, на *ticket_flights.ticket_no* *tickets.ticket_no*; поля *ticket_no, flight_id* таблицы

boarding_passes являются внешними ключами для *ticket_flights.ticket_no, ticket_flights.flight_id* соответственно.

4.1.8. Таблица *bookings.tickets* (билеты).

Билет имеет уникальный номер (*ticket_no*, естественный ключ таблицы), состоящий из 13 цифр. 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. Ограничения внешнего ключа: на t*ickets.book_ref* ссылается поле *bookings.book_ref*. Поле *tickets.ticket_no* ссылается на *ticket_flights.ticket_no*.

4.2.Материализованное представление bookings.routes

Таблица рейсов содержит избыточность: из неё можно было бы выделить информацию о маршруте (номер рейса, аэропорты отправления и назначения), которая не зависит от конкретных дат рейсов. Именно такая информация и составляет материализованное представление routes.

4.3.Представления

4.3.1. Представление *bookings.flights_v*

Над таблицей *flights* создано представление *flights_v*, содержащее дополнительную 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 полёта (*scheduled_duration, actual_duration*).

4.3.2. Представления *airports, aircrafts*

Представления, расшифровывающие данные городов и находящихся в них аэропортов, моделей воздушных судов применительно к текущей локали созданы над таблицами *airports_data* (представление *airports*) и *aircrafts_data* (представление *aircrafts*).

4.4.Бизнес-задачи, которые можно решить, используя базу данных.

Основная задача, которая может быть решена с использованием демонстрационной базы данных, это использование базы данных в учебных целях. В статье разработчика базы данных описан процесс и принципы подготовки информации для наполнения базы и формулирован ряд вопросов, на которые могут быть получены ответы с использованием SQL-запросов.

Следующая задача, которая может быть решения с использованием демонстрационной базы данных, это определение оптимальной системы рейсов и используемых для их выполнения воздушных судов. Оптимальная схема рейсов должна обеспечивать перевозку необходимого количества пассажиров между аэропортами с минимальным количеством перелётов, удобной стыковкой рейсов и максимальной загрузкой используемых воздушных судов.

Интересная задача, связанная с предыдущей, — повышение доходности деятельности перевозчика за счёт повышения затрат на приобретение билетов каждым отдельным пассажиром. Необходимость перемещения пассажиров между аэропортами обусловлена объективными причинами, не зависящими от действий авиакомпаний. В этих условиях авиаперевозчик имеет возможность предложить схему рейсов, вынуждающих пассажира совершать максимальное количество перелётов и приобретать билеты на каждый из них. Приобретение большего числа билетов повышает доход авиакомпании; с другой стороны, возрастают затраты на перевозку и уменьшается пассажиропоток вследствие использования некоторой частью пассажиров других видов транспорта.

Выбор оптимальной схемы перевозок как раз и может быть осуществлён с использованием демонстрационной базы данных.

5. Список SQL-запросов (файл eremeev.sql) с описанием логики их выполнения

1.В каких городах больше одного аэропорта?

```
select apt.city ->> lang() as "город", apt.cnt as "количество аэропортов",
        string_agg (apt.airport_name ->> lang(), ', ' order by apt.airport_name ->>
        lang()) as "аэропроты"
from
        (select         distinct  a.city,
                        count (a.airport_code) over (partition by a.city) as cnt,
                        a.airport_name
        from airports_data a) as apt
where apt.cnt > 1
group by apt.city ->> lang(), apt.cnt;
```

Подзапрос связывает город, число находящихся в нём аэропортов (определяемое при помощи оконной функции) и названия этих аэропортов. В основном запросе осуществляется отбор городов, в которых число аэропортов более одного и форматируется итоговая таблица для лучшего восприятия заказчиком.

2. В каких аэропортах есть рейсы, выполняемые самолётом с максимальной дальностью перелёта?

На первый взгляд кажется, что нет необходимости отображать раздельно аэропорты вылета и аэропорты прибытия. Действительно, если это аэропорт базирования, то вылетевший из него самолёт рано или поздно туда вернётся. А если это промежуточный аэропорт - прибывший в него самолёт когда-нибудь оттуда вылетит. Но вдруг есть или может быть создана какая-либо специфическая схема авиаперевозок, при которой осуществляется перелёт (в терминах используемой базы данных «рейс») без пассажиров? Потому, например, что взлёт значительно опаснее посадки или наоборот.

Таким образом, в подзапросе реализован выбор самолёта с максимальной дальностью перелёта, далее осуществлён отбор аэропортов (отдельно вылета и прибытия), в которых имеются рейсы, выполняемые с использованием этого самолёта, и результаты объединены в одну таблицу для удобства восприятия потребителем информации.

```
select ad.airport_name ->> lang() as "название аэропорта", 'аэропорт вылета' as
        "статус аэропорта"
from airports_data ad
where ad.airport_code in
        (select f.departure_airport
        from flights f
        where f.aircraft_code =
                (select ad.aircraft_code
                from aircrafts_data ad
                where ad."range" = (select max(ad2."range") from aircrafts_data ad2)))
union all
select ad.airport_name ->> lang() as "название аэропорта", 'аэропорт прибытия' as
        "статус аэропорта"
from airports_data ad
where ad.airport_code in
        (select f.arrival_airport
        from flights f
        where f.aircraft_code =
                (select ad.aircraft_code
                from aircrafts_data ad
                where ad."range" = (select max(ad2."range") from aircrafts_data ad2)));
```

3. Вывести 10 рейсов с максимальным временем задержки вылета.

В подзапросе реализовано определение времени задержки каждого рейса с заменой значения *null* на «0» и ранжирование результатов по убыванию (*desc*). В основном запросе осуществлена сортировка номеров ранга по убыванию и отбор 10 значений «с головы» полученного списка.

```
select res."date" as "дата рейса", res.flight_no as "номер рейса", res.t as "время
        задержки вылета"
from
        (select         f.scheduled_departure::date as "date", f.flight_no, coalesce
                        (f.actual_departure - f.scheduled_departure, '0') as t,  rank () over
                        (order by coalesce (f.actual_departure - f.scheduled_departure, '0')
                        desc)
        from flights f
        order by rank
        limit 10)
        as res;
```

4. Были ли брони, по которым не были получены посадочные талоны?

Как известно, в одно бронирование может включаться более одного билета и организация авиаперевозок не предполагает одновременного выполнения рейсов по всем этим билетам. Таким образом, помимо бронирований, в составе которых находятся *только* билеты, по которым посадочные талоны были выданы и бронирований, в составе

которых находятся *только* билеты, по которым посадочные талоны выданы не были, могут быть бронирования, в составе которых есть как те, так и другие билеты.

В подзапросе каждому номеру бронирования присваивается статус в зависимости от наличия в составе этого бронирования соответствующих билетов. *Left join* использован, поскольку необходимо отобрать все записи из таблицы *boarding_passes* и только те записи из таблицы *bookings*, для которых найдётся соответствие. В основном запросе подсчитывается число бронирований по каждому статусу.

```
select distinct res.status as "выдача посадочных талонов",
        count (*) over (partition by res.status) as "количество бронирований"
from
        (select distinct b.book_ref,
                case  when count (t.ticket_no) over (partition by b.book_ref) <> 0
                              and count (t.ticket_no) over (partition by b.book_ref) =
                              count (bp.boarding_no) over (partition by b.book_ref)
                              then 'выданы по всем билетам'
                      when count (t.ticket_no) over (partition by b.book_ref) <> 0
                              and count (t.ticket_no) over (partition by b.book_ref) <>
                              count (bp.boarding_no) over (partition by b.book_ref)
                              and count (bp.boarding_no) over (partition by b.book_ref)
                              <> 0
                              then 'выданы не по всем билетам'
                      else    'не выданы по всем билетам'
                end "status"
        from bookings b
        join tickets t on t.book_ref = b.book_ref
        left join boarding_passes bp using (ticket_no)) as res;
```

5. Найдите количество свободных мест для каждого рейса, их % отношение к общему количеству мест в самолёте. 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 течение дня.

Подзапросом устанавливаем количество мест в самолёте каждого типа, соединяем с использованием join все необходимые таблицы, а также результаты выполнения подзапроса и при помощи несложных математических операций и оконной функции sum с сортировкой по дате вылета определяем нужные данные.

```
select      ad.airport_name ->> lang() as "аэропрот вылета",
            f.actual_departure::date as "дата вылета",
            f.flight_no as "№ рейса",
            cnt_seats.cnt_s - count (bp.seat_no) as "количество свободных мест",
            round ((cnt_seats.cnt_s - count (bp.seat_no)) * 100.0 / cnt_seats.cnt_s, 1)
            as "доля свободных мест, %",
            sum (count (bp.seat_no)) over (partition by f.departure_airport,
            f.actual_departure::date order by f.actual_departure) as "вылетело
            пассажиров за день"
from flights f
join ticket_flights tf using (flight_id)
join boarding_passes bp using (ticket_no, flight_id)
join airports_data ad on ad.airport_code = f.departure_airport
join    (select s.aircraft_code, count (s.seat_no) as cnt_s
        from seats s
        group by s.aircraft_code) as cnt_seats
        on cnt_seats.aircraft_code = f.aircraft_code
group by ad.airport_name, f.actual_departure,  cnt_seats.cnt_s, f.flight_no,
f.departure_airport
order by ad.airport_name ->> lang(), f.actual_departure::date;
```

6. Найдите процентное соотношение перелётов по типам самолётов от общего количества.

В CTE получим ответ на поставленный вопрос, округлив результат до двух знаков после запятой. Основным запросом оформим результат в форме, удобной для восприятия потребителем информации.

```
with ratio as
        (select distinct    f.aircraft_code,
                            round ((count (f.flight_id) over (partition by f.aircraft_code))
                            *100.0 / (select count (*) from flights), 2)
        from flights f)
select  ad.model ->> lang() as "модель самолёта", ratio."round" as "доля перелётов, %"
from ratio
join aircrafts_data ad using (aircraft_code)
order by ratio."round" desc;
```

7. Были ли города, в которые можно добраться бизнес - классом дешевле, чем эконом-классом в рамках перелёта?

При помощи CTE установим и объединим в один столбец данные о минимальной стоимости билета бизнес-класса и максимальной стоимости билета эконом-класса обслуживания. Поскольку после этого на каждый перелёт будет приходиться столько строк, сколько в рамках этого перелёта предлагается тарифов, я не придумал ничего лучше, как с использованием оконных функций max и min сопоставить каждому перелёту единственную минимальную цену в рамках бизнес-класса и единственную же максимальную цену в рамках эконом-класса.

И, наконец, основной запрос обеспечивает отбор перелётов, для которых максимальная цена эконом-класса выше минимальной цены бизнес-класса и минимальная цена бизнес-класса отлична от 0 (т.е. на рейсе вообще есть бизнес-класс).

```
select res.flight_id as "ID перелёта", res.flight_no as "№ рейса", res.d_f as "дата рейса",
        res.city as "город прибытия"
from
        (with amnt as
                (select distinct        tf.flight_id, f.flight_no, f.scheduled_arrival::date as d_f,
                                        ad.city ->> lang() as city, tf.fare_conditions, tf.amount,
                                        coalesce (max (tf.amount) filter
                                        (where tf.fare_conditions = 'Economy') over (partition
                                        by tf.flight_id, city, tf.fare_conditions), '0') + coalesce
                                        (min (tf.amount) filter (where tf.fare_conditions =
                                        'Business') over (partition by tf.flight_id, city,
                                        tf.fare_conditions), '0') as max_amount
                from ticket_flights tf
                join flights f using (flight_id)
                join airports_data ad on f.arrival_airport = ad.airport_code)
        select distinct         amnt.flight_id, amnt.flight_no, amnt.d_f, amnt.city, coalesce
                                (min (amnt.max_amount) filter (where amnt.fare_conditions
                                = 'Business') over   partition by amnt.flight_id, amnt.city),
                                '0') as max_bus, coalesce (max  (amnt.max_amount) filter
                                (where amnt.fare_conditions = 'Economy') over (partition by
                                amnt.flight_id, amnt.city), '0') as max_eco
        from amnt) as res
where res.max_bus < res.max_eco and res.max_bus <> 0;
```

Судя по тому, что запрос возвращает пустую таблицу, таких перелётов нет.

8. Между какими городами нет прямых рейсов?

Декартовым произведением в from определяем все возможные пары городов, с использованием except исключаем сначала пары городов, между которыми есть рейсы в прямом, а затем в обратном направлении. Результат оборачиваем в CTE и исключаем дубли. Вариант критерия отбора *twin.city_1 < twin.city_2*, позволяющий исключить зеркальные пары, намеренно не рассматривается ради универсализации запроса, поскольку отсутствие рейса в прямом направлении не гарантирует отсутствия рейса в обратном.

```
with twin as
        (select *
        from
        (select ad.city ->> lang()  as city_1 from airports_data ad) as city_1,
        (select ad2.city ->> lang()  as city_2 from airports_data ad2) as city_2
        except
        select distinct ad.city ->> lang() as city_1, ad2.city ->> lang() as city_2
```

```
        from flights f
        join airports_data ad on f.departure_airport = ad.airport_code
        join airports_data ad2 on f.arrival_airport = ad2.airport_code
        except
        select distinct ad2.city ->> lang() as city_1, ad.city ->> lang() as city_2
        from flights f
        join airports_data ad on f.departure_airport = ad.airport_code
        join airports_data ad2 on f.arrival_airport = ad2.airport_code)
select twin.city_1 as "город 1", twin.city_2 as "город 2"
from twin
where twin.city_1 <> twin.city_2
order by city_1, city_2;
```

9. Вычислите расстояние между аэропортами, связанными прямыми рейсами, сравните с допустимой максимальной дальностью перелётов в самолётах, обслуживающих эти рейсы.

В CTE сформируем таблицу с необходимыми исходными данными: модель самолёта, дальность его полёта, название города и аэропорта вылета и назначения, рассчитанное расстояние между аэропортами.

В основном запросе придадим выходным данным форму, удобную для восприятия потребителем информации и при помощи case сравним расстояние между аэропортами и максимальной дальностью полёта самолётов, обслуживающих соответствующие рейсы.

Зеркальные пары аэропортов не исключаем ради универсализации запроса, поскольку рейсы в прямом и обратном направлениях, а принципе, могут быть выполнены различными типами воздушных судов.

```
with table_flights as
        (select distinct      ac.model ->> lang() as model, ac."range", ad.city ->> lang()
        as d_city, ad.airport_name ->> lang() as d_airport, ad2.city ->> lang() as a_city,
        ad2.airport_name ->> lang () as a_airport, round ((6371 * radians (acosd (sind
        (ad.coordinates [1]) * sind (ad2.coordinates [1]) + cosd (ad.coordinates [1]) *
        cosd (ad2.coordinates [1]) * cosd (ad.coordinates [0] -
        ad2.coordinates[0])))::numeric, 1) as distance
        from flights f
        join airports_data ad on f.departure_airport = ad.airport_code
        join airports_data ad2 on f.arrival_airport = ad2.airport_code
        join aircrafts_data ac on f.aircraft_code = ac.aircraft_code)
select table_flights.model as "модель самолёта", table_flights."range" as "дальность
полёта", table_flights.d_city as "город вылета", table_flights.d_airport as "аэропрот
вылета", a_city as "город назначения", table_flights.a_airport as "аэропорт назначения",
table_flights.distance as "дальность перелёта",
        case  when table_flights."range" - table_flights.distance > 1000 then 'прекрасно
              долетит'
              when table_flights."range" - table_flights.distance > 500 then 'хорошо
              долетит'
```

```
            when table_flights."range" - table_flights.distance > 50 then 'нормально
            долетит'
            when table_flights."range" - table_flights.distance > 10 then 'сможет
            долететь'
            when table_flights."range" - table_flights.distance > 0.00001 then 'почти
            долетит' .
            else 'шансов нет'
        end as "шанс долететь",
table_flights."range" - table_flights.distance as "запас полёта"
from table_flights;
```