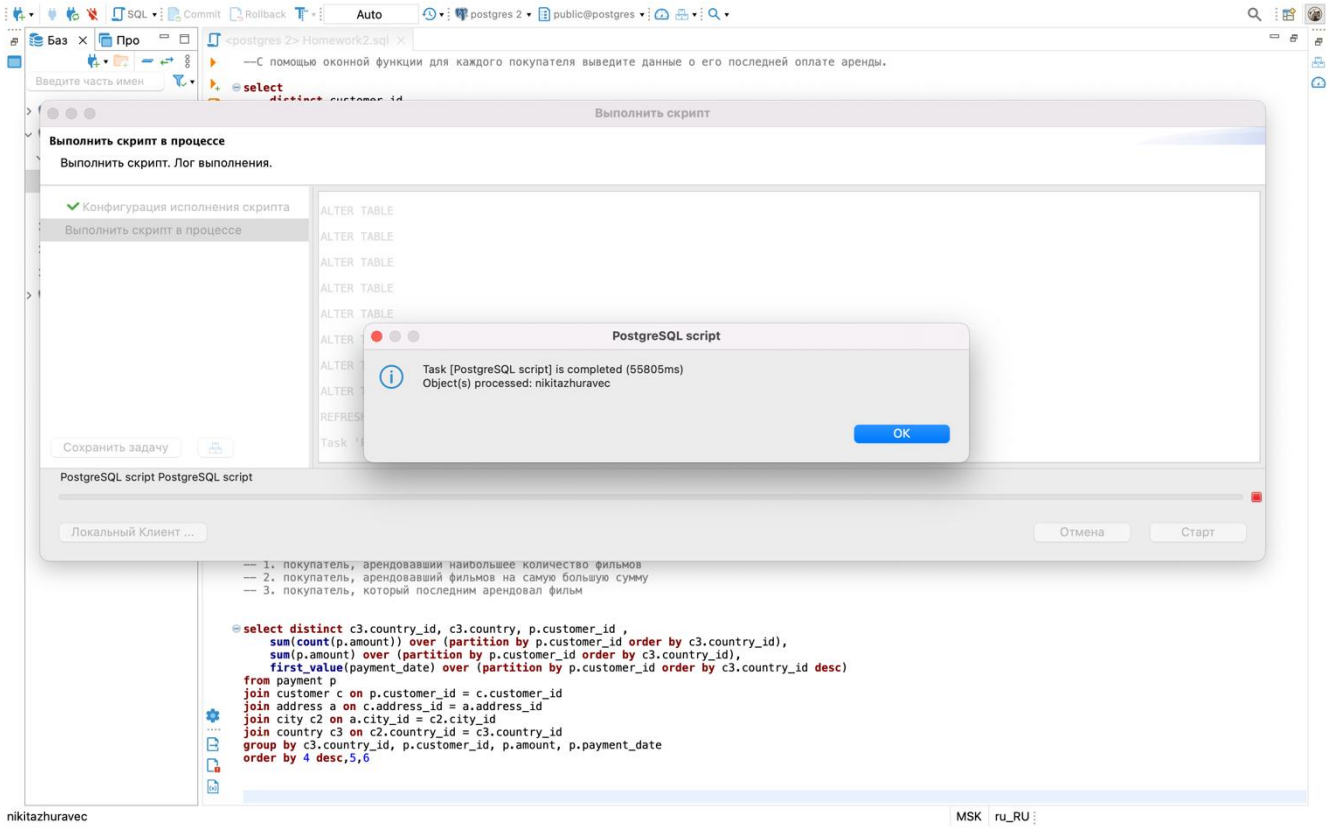
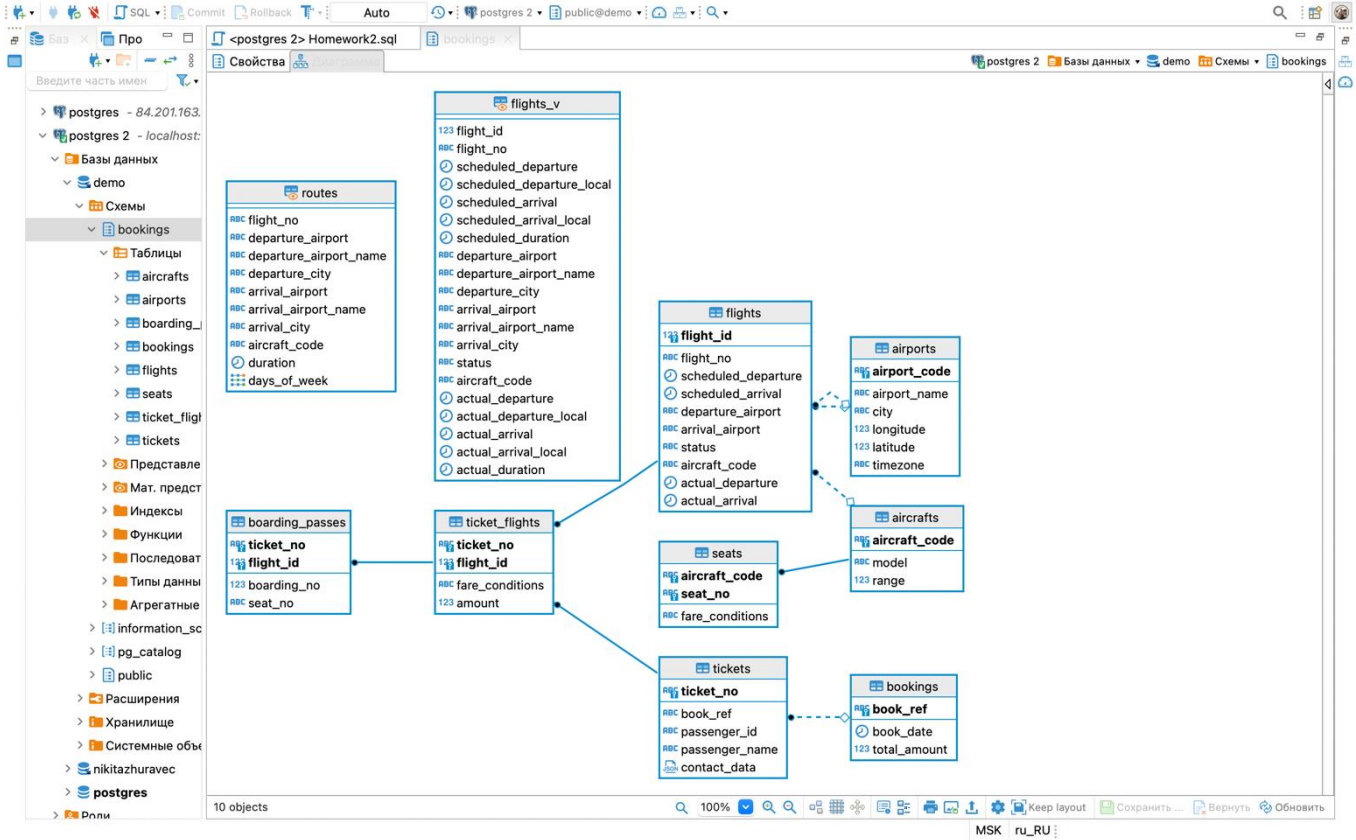
Итоговая работа "SQL и получение данных"

1. В работе использовался локальный тип подключения.



2. Скриншот ER-диаграммы из DBeaver`a.



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

Таблицы:

- bookings.aircrafts
- bookings.airports
- bookings.boarding_passes
- bookings.bookings
- bookings.flights
- bookings.seats
- bookings.ticket_flights
- bookings.tickets

Представления:

- Представление "bookings.flights_v"
- Материализованное представление bookings.routes

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

Таблица **bookings.aircrafts**

Каждая модель воздушного судна идентифицируется своим трехзначным кодом (aircraft_code). Указывается также название модели (model) и максимальная дальность полета в километрах (range).

1. Индексы:
   - **PRIMARY KEY** btree(aircraft_code).
2. Ограничения:
   - aircraft_code, model, range **NOT NULL**;
   - aircrafts_range_check > 0;
3. Ссылки:
   - Таблица flights **FOREIGN KEY**(aircraft_code) **REFERENCES** aircrafts(aircraft_code);
   - Таблица seats **FOREIGN KEY**(aircraft_code) **REFERENCES** aircrafts(aircraft_code) **ON DELETE CASCADE.**

Таблица **bookings.airports**

Аэропорт идентифицируется трехбуквенным кодом (airport_code) и имеет свое имя (airport_name). 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).

1. Индексы:
   - **PRIMARY KEY** btree(airport_code).
2. Ограничения:
   - airport_code, airport_name, city, longitude, latitude, timezone **NOT NULL**.
3. Ссылки:
   - Таблица flights **FOREIGN KEY**(arrival_airport) **REFERENCES** aiports(airport_code);
   - Таблица flights **FOREIGN KEY**(departure_airport) **REFERENCES** aiports(airport_code).

Таблица **bookings.boarding_passes**

При регистрации на рейс, которая возможна за сутки до плановой даты отправления, пассажиру выдается посадочный талон. Он идентифицируется также, как и перелет — 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).

1. Индексы:
    - **PRIMARY KEY** btree(ticket_no, flight_id**) CONSTRAINT;**
    - **UNIQUE KEY** btree(flight_id, boarding_no) **CONSTRAINT;**
    - **UNIQUE KEY** btree(flight_id, seat_no) **CONSTRAINT.**
2. Ограничения:
    - ticket_no, flight_id, boarding_no, seat_no **NOT NULL**;
3. Ограничения внешнего ключа:
    - **FOREIGN KEY**(ticket_no,flight_id) **REFERENCES** ticket_flights(ticket_no,flight_id).

Таблица **bookings.bookings**

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). Поле total_amount хранит общую стоимость включенных в бронирование перелетов всех пассажиров

1. Индексы:
    - **PRIMARY KEY** btree(book_ref**);**
2. Ограничения:
    - book_ref, book_date, total_amount **NOT NULL**;
3. Ссылки:
    - Таблица tickets **FOREIGN KEY** (book_ref) **REFERENCES** bookings(book_ref);

Таблица **bookings. flights**

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 (scheduled_arrival). Реальные время вылета (actual_departure) и прибытия (actual_arrival) могут отличаться: обычно не сильно, но иногда и на несколько часов, если рейс задержан.

1. Индексы:
    - **PRIMARY KEY** btree(flight_id**);**
    - **UNIQUE KEY** btree(flight_no, scheduled_departure) **CONSTRAINT.**
2. Ограничения:
    - flight_id, flight_no, scheduled_departure, scheduled_arrival, departure_airport, arrival_airport, status, aircraft_code, actual_departure, actual_arrival **NOT NULL**;
    - **CHECK** (scheduled_arrival > scheduled_departure);
    - **CHECK** (((actual_arrival **IS NULL**) **OR** ((actual_departure **IS NOT NULL**) **AND** (actual_arrival **IS NOT NULL**) **AND** (actual_arrival > actual_departure))));
    - **CHECK** (((status)::**text** = **ANY** (**ARRAY**[('On Time'::**character varying**)::**text**, ('Delayed'::**character varying**)::**text**, ('Departed'::**character varying**)::**text**, ('Arrived'::**character varying**)::**text**, ('Scheduled'::**character varying**)::**text**, ('Cancelled'::**character varying**)::**text**]))).
3. Ссылки:
    - Таблица ticket_flights **FOREIGN KEY** (flight_id) **REFERENCES** flights (flight_id).
4. Ограничения внешнего ключа:
    - **FOREIGN KEY** (aircraft_code) **REFERENCES** aircrafts(aircraft_code);
    - **FOREIGN KEY** (arrival_airport) **REFERENCES** airports(airport_code);

- **FOREIGN KEY** (departure_airport) **REFERENCES** airports(airport_code).

Таблица **bookings.seats**

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.

1. Индексы:
   - **PRIMARY KEY** btree(aircraft_code, seat_no**) CONSTRAINT.**
2. Ограничения:
   - aircraft_code, seat_no, fare_conditions **NOT NULL**;
   - **CHECK (((fare_conditions)::text = ANY (ARRAY[('Economy'::character varying)::text, ('Comfort'::character varying)::text, ('Business'::character varying)::text]))).**
3. Ограничения внешнего ключа:
   - **FOREIGN KEY** (aircraft_code) **REFERENCES aircrafts(aircraft_code) ON DELETE CASCADE**

Таблица **bookings.ticket_flights**

Перелет соединяет билет с рейсом и идентифицируется их номерами. Для каждого перелета указываются его стоимость (amount) и класс обслуживания (fare_conditions).

1. Индексы:
   - **PRIMARY KEY** btree(ticket_no, flight_id**) CONSTRAINT.**
2. Ограничения:
   - ticket_no, flight_id, fare_conditions, amount **NOT NULL**;
   - **CHECK** ((amount >= (0)::numeric));
   - **CHECK** (((fare_conditions)::**text** = **ANY** (**ARRAY**[('Economy'::**character varying**)::**text**, ('Comfort'::**character varying**)::**text**, ('Business'::**character varying**)::**text**])));
3. Ссылки:
   - Таблица boarding_passes **FOREIGN KEY** (ticket_no, flight_id) **REFERENCES** ticket_flights(ticket_no, flight_id)
4. Ограничения внешнего ключа:
   - **FOREIGN KEY** (flight_id) **REFERENCES** flights(flight_id);
   - **FOREIGN KEY** (ticket_no) **REFERENCES** bookings.tickets(ticket_no).

Таблица **bookings.tickets**

Билет имеет уникальный номер (ticket_no), 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).

1. Индексы:
   - **PRIMARY KEY** btree(ticket_no**).**
2. Ограничения:
   - ticket_no, book_ref, passenger_id, passenger_name, contact_data **NOT NULL**.
3. Ссылки:
   - Таблица ticket_flights **FOREIGN KEY** (ticket_no) **REFERENCES** tickets (ticket_no)
4. Ограничения внешнего ключа:
   - **FOREIGN KEY (book_ref) REFERENCES bookings(book_ref).**

Бизнес задачи, которые можно решить при помощи БД

1. Оптимизировать отношение максимально возможной дальности перелета и реальной дальность перелета самолетов.
2. Открытие новых маршрутов.
3. Оптимизация цены на стоимость билетов.
4. Выявление и оптимизация заполняемости мест самолетов.
5. Выявление задержек рейсов.

Список SQL запросов из приложения №2 с описанием логики их выполнения.

№1 В каких городах больше одного аэропорта?

```
select city as "Города"
from (
	select city, count(airport_name) as count
	from airports a
	group by city) t
where count > 1
```

Логика:
При помощи подзапроса и функции count производим подсчет количества аэропортов в каждом городе. Оператором where оставляем на вывод только те города, где больше 1 аэропорта.

№2 В каких аэропортах есть рейсы, выполняемые самолетом с максимальной дальностью перелета?

```
select a2.airport_name as "Аэропорты"
from (
	select aircraft_code, row_number() over (order by "range" desc) as r_w
	from aircrafts a) t
left join flights f on t.aircraft_code = f.aircraft_code
left join airports a2 on f.departure_airport = a2.airport_code
where r_w = 1
group by a2.airport_name
```

Логика:
При помощи функции row_number сортируем самолеты по максимальной дальности перелета. Оператором where оставляем только самолет с максимальной дальностью перелета. При помощи оператора join присоединяем таблицы flights и airports для того, чтобы вывести в результат аэропорты. Группируем по имени аэропорта.

№3 Вывести 10 рейсов с максимальным временем задержки вылета

```
select flight_no  as "Рейсы"
from(
	select flight_no, (actual_departure - scheduled_departure)
```

```
        from flights
        where actual_departure is not null and actual_departure > scheduled_departure
        order by 2 desc) t
limit 10
```

Логика:
Используя подзапрос, для каждого рейса, где фактическое время вылета позже, чем планируемое находим разницу. При помощи оператора order by сортируем по времени задержки в порядке от большего к меньшему. При помощи оператора limit оставляем на вывод 10 строк.

№4 Были ли брони, по которым не были получены посадочные талоны?

```
select b.book_ref
from bookings b
full outer join tickets t on b.book_ref = t.book_ref
left join boarding_passes bp on bp.ticket_no = t.ticket_no
where bp.ticket_no is null or bp.flight_id is null */
```

Логика:
Присоединяем к таблице броней (bookings) таблицу с билетами (tickets) при помощи оператора full outer join, для того чтобы сохранить все данные из этих таблиц. Также присоединяем таблицу boarding_passes в которой содержатся данные о пасадочных талонах. Оператором where оставляем только те брони, где не были получени посадочные талоны.

№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.

```
select res.flight_id, res.actual_departure, res.departure_airport, q.count - res.count as "Свободные места", round((1- (res.count::numeric / q.count::numeric))*100 , 2) as "Свободные места, %", res.count as "Вывезено", res.sum as "Накопление"
from (
        select f.flight_id, f.actual_departure, a.aircraft_code , f.departure_airport, count(tf.ticket_no), sum(count(tf.ticket_no)) over (partition by f.actual_departure::date, f.departure_airport order by f.actual_departure)
        from flights f
        left join ticket_flights tf on f.flight_id = tf.flight_id
        join aircrafts a on a.aircraft_code = f.aircraft_code
        where f.actual_departure is not null
        group by f.flight_id, f.actual_departure, f.arrival_airport, a.aircraft_code) res
left join (
select s.aircraft_code, count(s.seat_no)
from seats s
group by s.aircraft_code) q on res.aircraft_code = q.aircraft_code
```

Логика:
Используя подзапрос считаем количество купленных билетов (ticket_no) и накопительный итог в рамках для каждого перелета. При помощи оператора where убираем несостоявшиеся перелеты. Обьединяем результат первого подзапроса со вторым, в котором считаем общее количество мест(seats) в каждом из перелетов. Выводим в результат запроса номер перелета, время вылета, аэропорт, количество свободных мест (разницу между общим количеством мест и количеством купленных билетов), процент свободных мест, количество занятых мест для каждого из вылетов и накопительный итог в рамках дня для каждого из аэропортов.

№6 Найдите процентное соотношение перелетов по типам самолетов от общего количества.

```
select distinct a.model,
	round((count(f.flight_id) over (partition by f.aircraft_code)::numeric / 
count(f.flight_id) over ())*100,2) || '%' as "Процентное соотношение"
from flights f
left join aircrafts a on f.aircraft_code = a.aircraft_code
where f.actual_departure is not null
```

Логика:
При помощи оператора left join объединяем таблицы flights и aircrafts. Оператором where убираем перелеты, где вылет не состоялся. Выводим результат для каждого типа самолета – процентное соотношение перелетов по типам самолетов (считаем количество перелетов каждым самолетом и делим на общее количество перелетов, после чего округляем).

№7 Были ли города, в которые можно добраться бизнес - классом дешевле, чем эконом-классом в рамках перелета?

```
with eco as (
	select f.flight_id, tf.amount
	from ticket_flights tf
	left join flights f on tf.flight_id = f.flight_id
	left join airports a on f.arrival_airport = a.airport_code
	where tf.fare_conditions = ‘Economy’),
bus as (
	select f.flight_id, tf.amount, a.city
	from ticket_flights tf
	left join flights f on tf.flight_id = f.flight_id
	left join airports a on f.arrival_airport = a.airport_code
	where tf.fare_conditions = ‘Business’)
select bus.city
from eco
join bus on eco.flight_id = bus.flight_id
where eco.amount > bus.amount
group by bus.city
```

Логика:
Используя CTE и оператор where разделяем перелеты бизнес-класс и эконом-класса. В каждом из них выводим данные по перелету и стоимости перелета при помощи обьединения таблиц оператором left join. Далее обьединяем эти CTE и сравниваем

стоимости бизнес и эконом классы по стоимости. Выводим города, в которые можно добраться бизнес-классом дешевле, чем эконом-классом.

№8 Между какими городами нет прямых рейсов?

```
CREATE VIEW cities as
SELECT a.city as dep, a2.city as arr
FROM airports a, airports a2
where a.city != a2.city

select *
from cities
except (
with dep as
(
select distinct f.flight_no, a.city
from flights f
left join airports a on f.departure_airport = a.airport_code
),
arr as
(
select distinct f.flight_no, a.city
from flights f
left join airports a on f.arrival_airport = a.airport_code
)
select dep.city, arr.city
from dep
left join arr on dep.flight_no = arr.flight_no)
order by 1,2
```

Логика:
Создаем представление в котором используя декартово произведение сопоставляем каждый город с каждым, для составления всех возможных перелетов между городами, исключая тот случай, когда приземление происходит в тот же город, откуда произошел взлет. При помощи оператора except и двух CTE исключаем уже существующие перелеты между городами.

№9 Вычислите расстояние между аэропортами, связанными прямыми рейсами, сравните с допустимой максимальной дальностью перелетов в самолетах,
* обслуживающих эти рейсы *

```
with coor1 as(
	select f.flight_no, f.departure_airport, f.aircraft_code, radians(a.longitude) as long, radians(a.latitude) as lat
	from airports a
	full outer join flights f on a.airport_code = f.departure_airport
	where f.actual_departure is not null),
coor2 as (
	select f.flight_no, f.arrival_airport, radians(a.longitude) as long, radians(a.latitude) as lat
	from airports a
	full outer join flights f on a.airport_code = f.arrival_airport
	where f.actual_arrival is not null)
```

```
select count(*),coor1.flight_no as "№", coor1.departure_airport as "Вылет",
coor2.arrival_airport as "Прибытие", (6371 * acos(sin(coor1.lat)*sin(coor2.lat) +
cos(coor1.lat)*cos(coor2.lat)*cos(coor1.long - coor2.long)))::int as "Расстояние", a2.range as
"Максимальное расс-е самолета",
        case
                when (6371 * (acos(sin(coor1.lat)*sin(coor2.lat) +
cos(coor1.lat)*cos(coor2.lat)*cos(coor1.long - coor2.long)))::int) < a2."range"::int then
'Максимальное расс-е самолета больше'
                else 'Максимальное расс-е самолета меньше'
        end as "Перелет"
from coor1
inner join coor2 on coor1.flight_no = coor2.flight_no
left join aircrafts a2 on coor1.aircraft_code = a2.aircraft_code
group by coor1.flight_no, coor1.departure_airport, coor2.arrival_airport, coor1.lat, coor1.long,
coor2.lat, coor2.long, a2.range
```

Логика:
Используя 2 CTE вычисляем широту и долготу для аэропортов в которых происходили вылет и посадка, номер перелета, а также модель самолета. Объединяем CTE, выводим номер рейса, аэропорт вылета и прибытия, при помощи формулы и уже известных нам данных вычисляем расстояние между аэропортами, и максимально возможную дальность полета самолета. При помощи оператора case сравниваем расстояние между аэропортами и максимально возможную дальность полета самолета. В случае, если максимальная дальность полета больше, то выводим – Максимальное расстояния самолета больше, иначе – Максимальное расстояние самолета меньше.