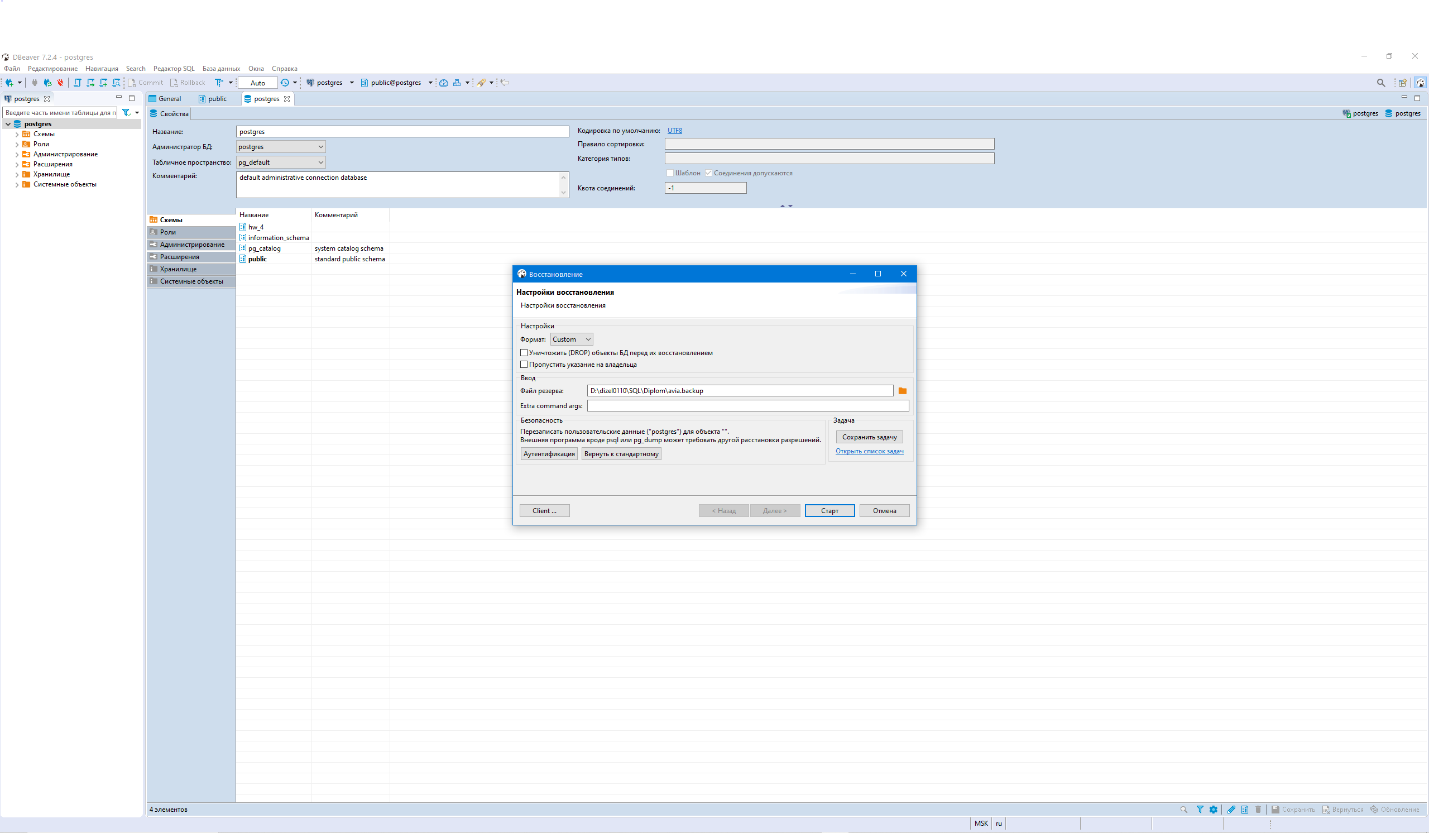
Итоговая работа по модулю “SQL и получение данных”

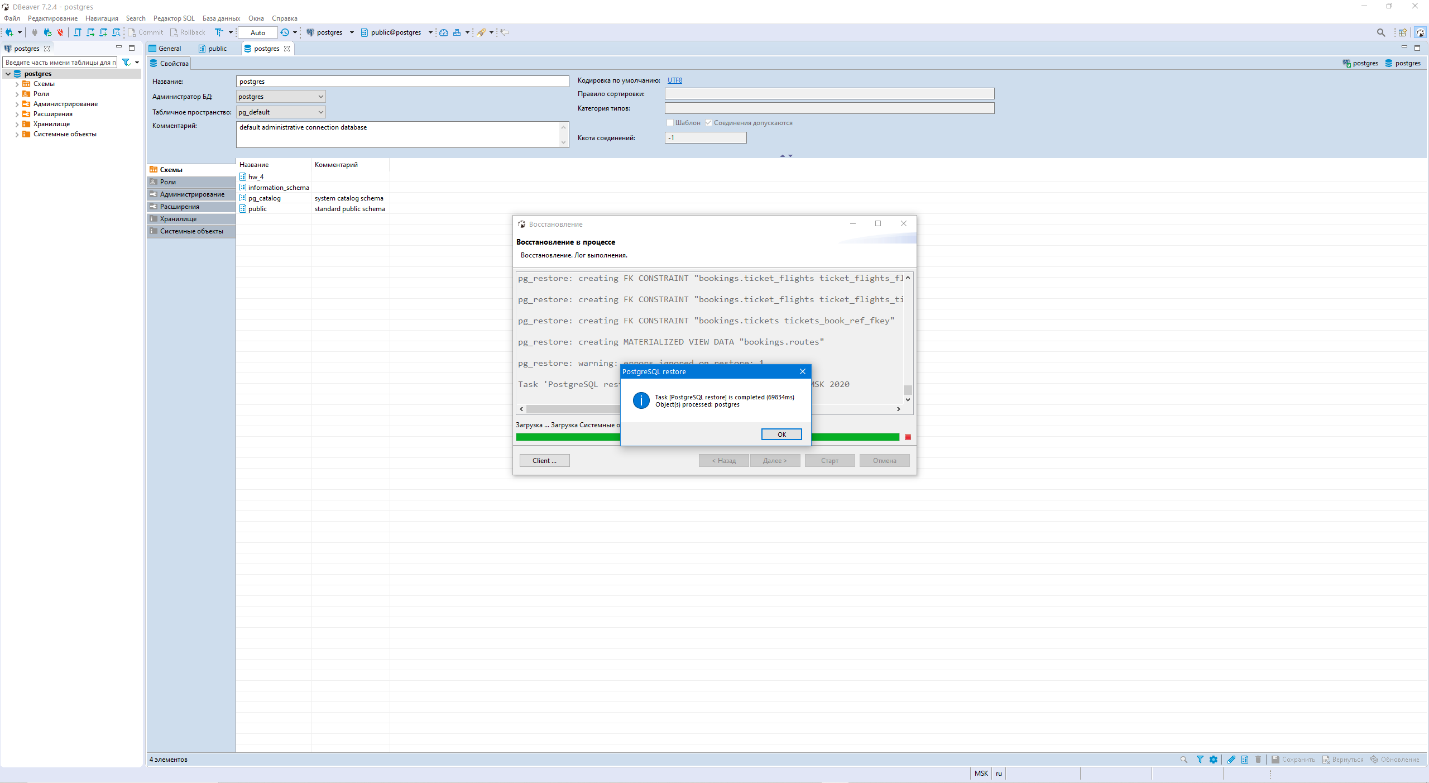
Группа SQL-23

Зеленин Дмитрий Викторович

1. В работе использовался локальный тип подключения.

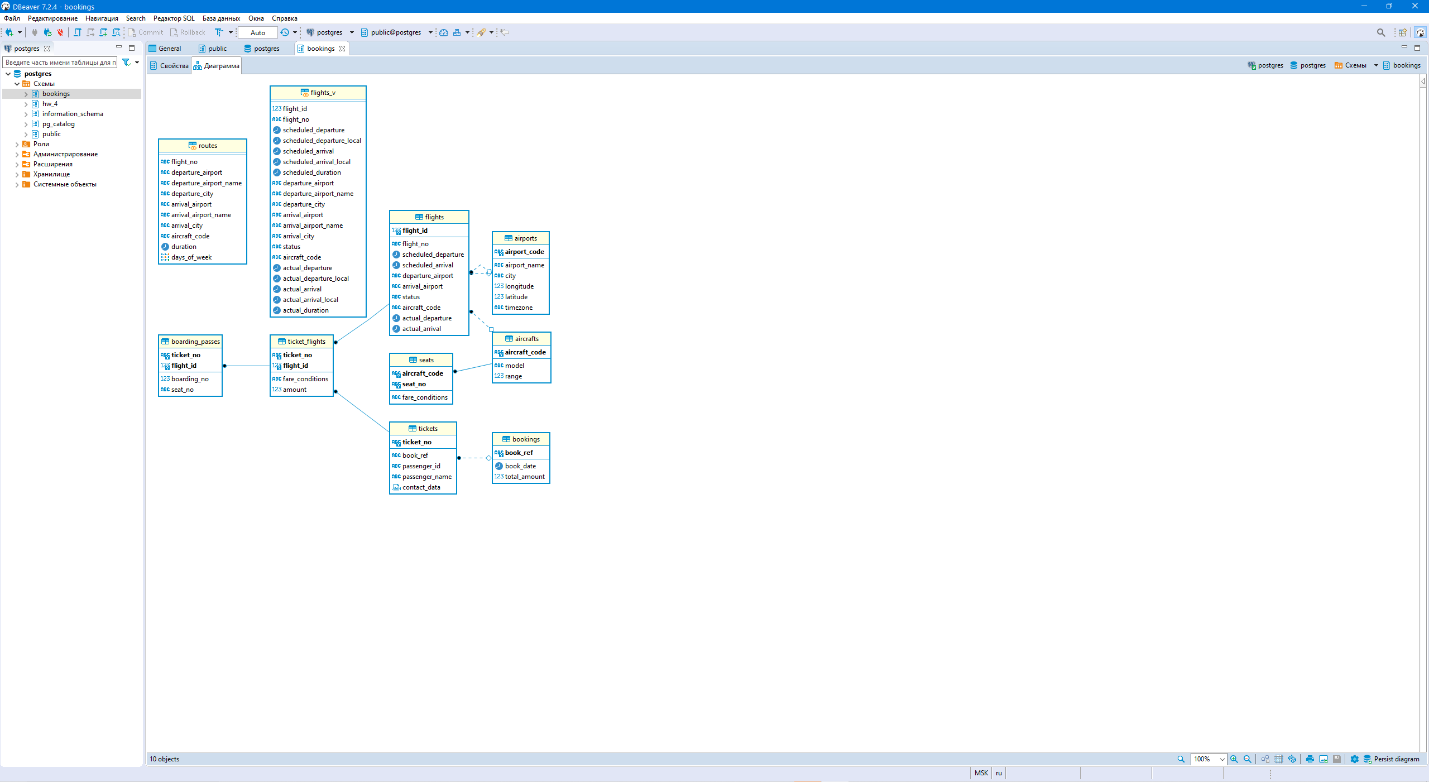
Скриншот успешного импорта или восстановления:

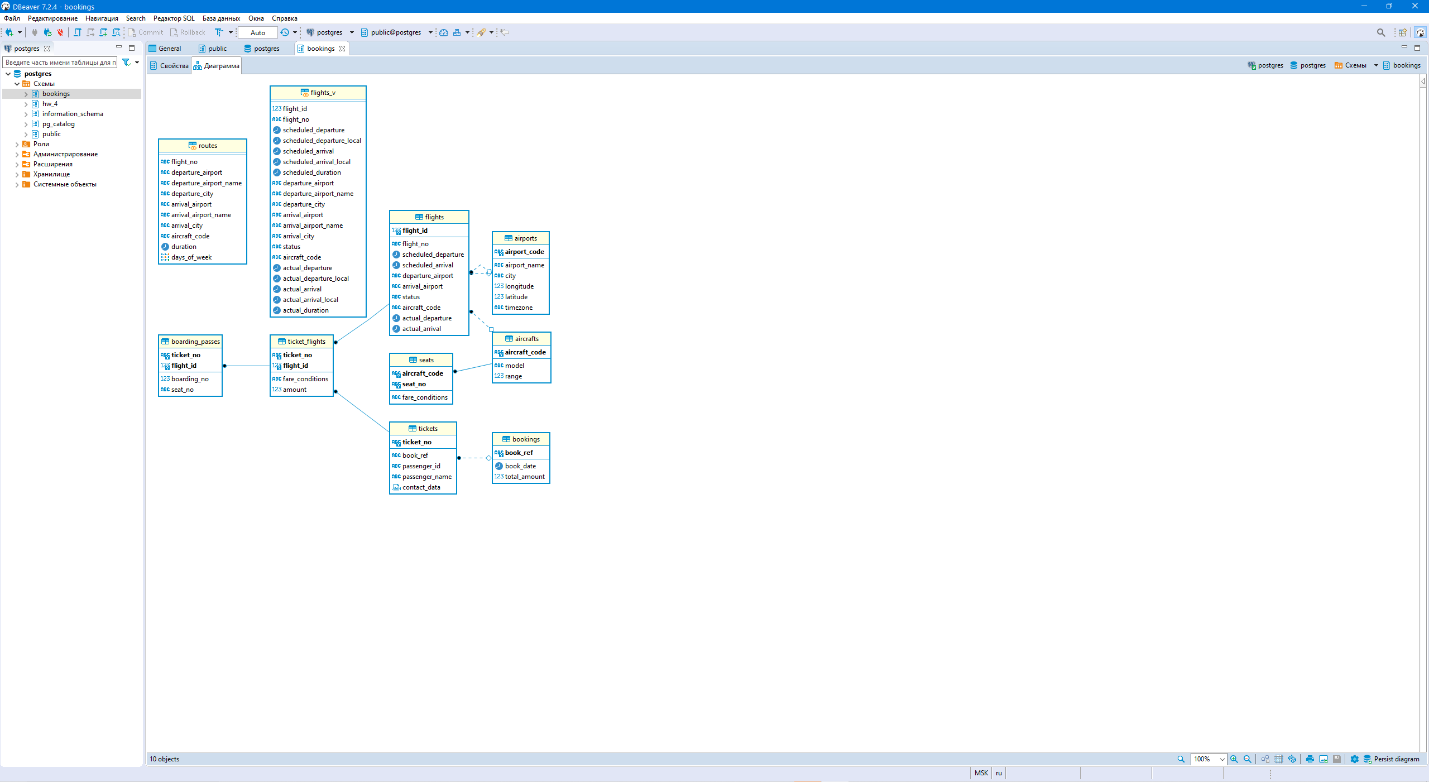




Восстановление заняло ~ 3 минуты 28 секунд.

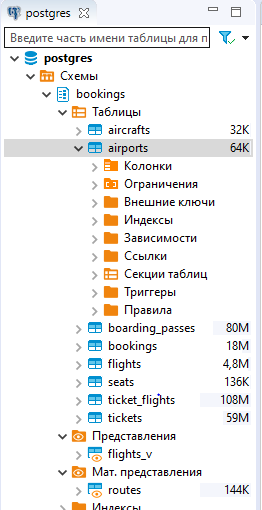
1. Скриншот ER-диаграммы из DBeaver`a согласно Вашего подключения.





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

Естественно, вся информация представлена в файле bookings.pdf. Можно добавить скрин из программы DBeaver, где представлены все таблицы и представления, в том числе и материализованные представления.



Конечно, можно написать код, который мы выполняли в домашних заданиях, находя все таблицы и ключи. Но уже из вышепредставленного рисунка видно, что у нас восемь таблиц, одно представление и одно материализованное представление. Это всё соответствует ER-диаграмме из пункта 2, естественно. Единственное, что из этого скриншота можно явно отличить представление flights\_v от материализованного представления routes.

1. Развернутый анализ БД

Основной сущностью здесь является бронирование (bookings).

В одно бронирование можно включить несколько пассажиров, каждому из которых выписывается отдельный *билет* (tickets). Как таковой пассажир не является отдельной сущностью: для простоты можно считать, что все пассажиры уникальны.

Билет включает один или несколько *перелетов* (ticket\_flights). Несколько перелетов могут включаться в билет в нескольких случаях:

1. Нет прямого рейса, соединяющего пункты отправления и назначения (полет с пересадками);
2. Взят билет «туда и обратно».

В схеме данных нет жесткого ограничения, но предполагается, что все билеты в одном бронировании имеют одинаковый набор перелетов.

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 выдается *посадочный талон* (boarding\_passes), в котором указано место в самолете. Пассажир может зарегистрироваться только на тот рейс, который есть у него в билете. Комбинация рейса и места в самолете должна быть уникальной, чтобы не допустить выдачу двух посадочных талонов на одно место.

Количество *мест* (seats) в самолете и их распределение по классам обслуживания зависит от модели *самолета* (aircrafts), выполняющего рейс. Предполагается, что каждая модель имеет только одну компоновку салона. Схема данных не контролирует, что места в посадочных талонах соответствуют имеющимся в самолете.

База довольна обширна, следует по мере выполнения задач расширять своё знание о базе, под конкретную задачу.

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

5.1 В каких городах больше одного аэропорта?

Воспользуемся таблицей bookings.airports. Нам понадобится данные о городе – столбец city, который может служить для того, чтобы определить аэропорты одного города. В качестве идентфикаторов аэропортов служит трехбуквенный код (airport\_code). Выведем город city, количество аэропортов в городе count(airport\_code) из нашей таблицы, группируя данные по городам, с количеством аэропортов в городе больше одного count(airport\_code) > 1. Результат можно упорядочить по трехбуквенному коду, по алфавиту.

Следующий SQL-запрос отвечает на поставленный вопрос:

**SELECT**

city,

**COUNT**(airport\_code) n\_air

**FROM**

bookings.airports a

**GROUP** **BY**

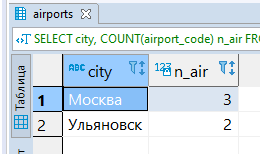
city

**HAVING**

**COUNT**(airport\_code) > 1

**ORDER** **BY**

city;



5.2 В каких аэропортах есть рейсы, выполняемые самолетом с максимальной дальностью перелета?

Здесь дополнительное ограничение, что самолёты совершают максимальную дальность перелёта целесообразно выполнить отдельным подзапросом:

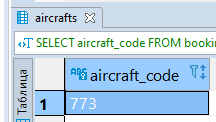
**SELECT**

aircraft\_code

**FROM**

bookings.aircrafts a

**WHERE** **range** = (**SELECT** **MAX**(**range**) **FROM** bookings.aircrafts);



После уже подзапрос можно сравнивать со всеми трехзначными кодами модели воздушного судна в условии where основного запроса в качестве дополнительного условия на максимальную дальность перелёта. И останется вывести данные в выходную таблицу с указанием нужной информации с учётом дополнительных условий: имя аэропорта, трехбуквенный код идентифицирующий аэропорт из таблицы аэропортов и модель самолёта из таблицы самолётов.

Соответствующий SQL-запрос:

**SELECT**

**DISTINCT** ap.airport\_name,

ap.airport\_code,

a.model

**FROM**

bookings.flights f

**LEFT** **JOIN** bookings.airports ap **ON**

f.departure\_airport = ap.airport\_code

**OR** f.arrival\_airport = ap.airport\_code

**LEFT** **JOIN** bookings.aircrafts a **ON**

f.aircraft\_code = a.aircraft\_code

**WHERE**

f.aircraft\_code = (

**SELECT**

aircraft\_code

**FROM**

bookings.aircrafts a

**WHERE**

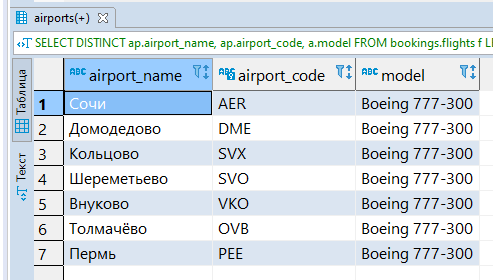
**range** = (

**SELECT**

**MAX**(**range**)

**FROM**

bookings.aircrafts));



5.3 Вывести 10 рейсов с максимальным временем задержки вылета

В этой задаче понадобится работа с временным типом данных. Для улучшенного восприятия переведём максимальное время задержки в стандартное отображение HH:MM:SS доступным для SQL-запросов способом. Работаем с таблицей flights, обращая чтобы значения в таблице по вылету не были f.actual\_departure IS NOT NULL и для сокращения работы запроса, не учитывая ненулевые значения задержки вылета (f.actual\_departure - f.scheduled\_departure)) != 0.

Соответствующий SQL-запрос:

**SELECT**

scheduled\_departure,

actual\_departure,

("delay\_in\_s"::**varchar**(24) || ' seconds')::**interval** **as** delay\_in\_HMS

**FROM**

(

**SELECT**

f.scheduled\_departure, f.actual\_departure, **ROUND**((**EXTRACT**(EPOCH **FROM** (f.actual\_departure - f.scheduled\_departure))) ::**numeric**, 1) **AS** delay\_in\_s

**FROM**

bookings.flights f

**WHERE**

f.actual\_departure **IS** **NOT** **NULL**

**AND** **EXTRACT**(EPOCH

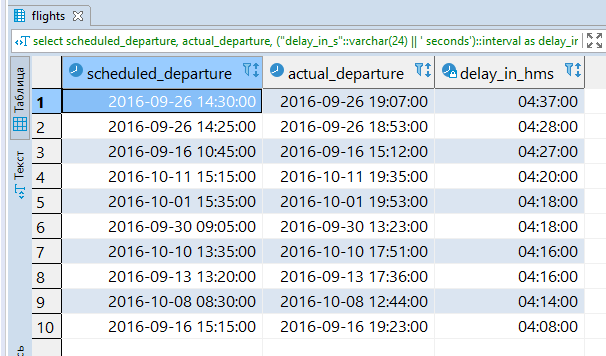
**FROM**

(f.actual\_departure - f.scheduled\_departure)) != 0

**ORDER** **BY**

delay\_in\_s **DESC**

**LIMIT** 10) sub;



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

Необходимые таблицы, естественно, это бронирования bookings и билеты tickets. Как говорится, начнём с дополнительного условия, остальное подтянется. Дополнительным условием будет условие, что посадочные талоны не были получены, фактически это описывается конструкцией bp.boarding\_no IS NULL.

Соответствующий SQL-запрос:

**SELECT**

**DISTINCT** b.book\_ref, bp.boarding\_no

**FROM**

bookings.bookings b

**RIGHT** **JOIN** bookings.tickets t **ON**

b.book\_ref = t.book\_ref

**LEFT** **JOIN** bookings.boarding\_passes bp **ON**

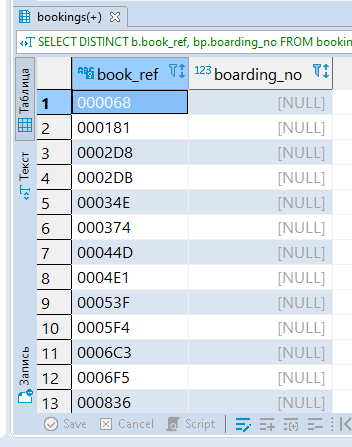
t.ticket\_no = bp.ticket\_no

**WHERE**

bp.boarding\_no **IS** **NULL**

**ORDER** **BY**

b.book\_ref;



5.5 Найдите свободные места для каждого рейса, их % отношение к общему количеству мест в самолете.

Добавьте столбец с накопительным итогом - суммарное количество вывезенных пассажиров из аэропорта за день. Т.е. в этом столбце должна отражаться сумма - сколько человек уже вылетело из данного аэропорта на этом или более ранних рейсах за день.

Следующим подзапросом

**SELECT**

aircraft\_code,

**COUNT**(aircraft\_code) t\_seats

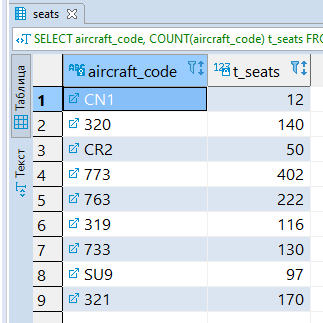
**FROM**

bookings.seats s

**GROUP** **BY**

aircraft\_code

получим таблицу, где указан код всех самолётов, участвующих в перелёте с указанием количества мест в салоне. Здесь воспользовались тем, что под конкретный код самолёта с уникальным (здесь я пишу в описании действий, поэтому не использую в запросе distinct 😊) названием представлены в другой колонке уникальные номера мест и можно сгруппировав и подсчитав количество строк с конкретным значением кода самолёта получить и количество мест для данного типа самолёта. Данный подзапрос можно использовать в качестве алиаса для дальнейших вычислений.



Также необходимые данные находятся в представлении "bookings.flights\_v"

SQL-запрос:

**SELECT**

fv.flight\_id,

fv.flight\_no,

fv.departure\_airport\_name dep\_name,

fv.arrival\_airport\_name arr\_name,

fv.aircraft\_code ac,

fv.actual\_departure act\_depart,

**COUNT** (fv.flight\_id) onboard,

ts.t\_seats, -- полученное в подзапросе общее количество мест в самолётах

ts.t\_seats - **COUNT** (fv.flight\_id) e\_seats,

**ROUND**(100 \* (1 - **COUNT**(fv.flight\_id) / ts.t\_seats::**numeric**), 2) realseats\_percent,

**SUM**(**COUNT** (fv.flight\_id))

**OVER**(**PARTITION** **BY** fv.departure\_airport\_name, **DATE**(fv.actual\_departure)

**ORDER** **BY** fv.actual\_departure

**ROWS** **BETWEEN** **UNBOUNDED** **PRECEDING** **AND** **CURRENT** **ROW**) sumpass\_day

-- The frame, ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW,

-- means that the window consists of the first row of the partition

-- and all the rows up to the current row.

-- Each calculation is done over a different set of rows.

-- For example, when performing the calculation for row 4, the rows 1 to 4 are used.

--

-- https://www.red-gate.com/simple-talk/sql/t-sql-programming/introduction-to-t-sql-window-functions/

**FROM** bookings.flights\_v fv

**LEFT** **JOIN** bookings.ticket\_flights tf **ON**

fv.flight\_id = tf.flight\_id

**RIGHT** **JOIN** bookings.boarding\_passes bp **ON**

tf.ticket\_no = bp.ticket\_no **AND** tf.flight\_id = bp.flight\_id

**LEFT** **JOIN** (

**SELECT**

aircraft\_code,

**COUNT**(aircraft\_code) t\_seats

**FROM**

bookings.seats s

**GROUP** **BY**

aircraft\_code) ts **ON**

fv.aircraft\_code = ts.aircraft\_code

**GROUP** **BY**

fv.flight\_id,

fv.flight\_no,

fv.departure\_airport\_name,

fv.arrival\_airport\_name,

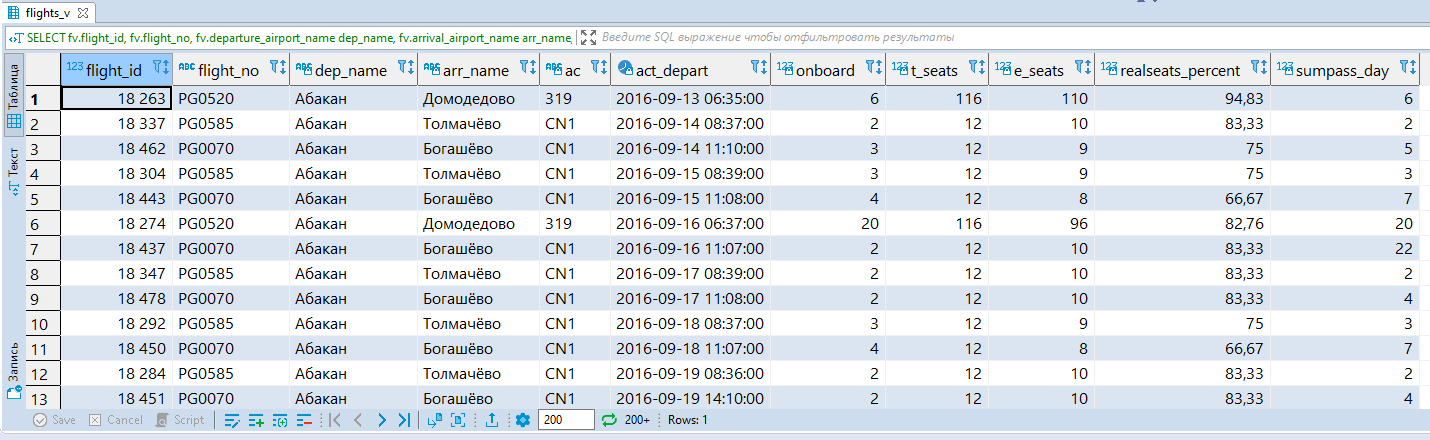
fv.aircraft\_code,

ts.t\_seats,

fv.actual\_departure

-- фактическое время вылета должно быть, то есть полёт состоялся.

**HAVING** fv.actual\_departure **IS** **NOT** **NULL**;



5.6 Найдите процентное соотношение перелетов по типам самолетов от общего количества.

Необходимо найти общее количество перелетов поделить к соотношению перелетов по типам перелетов и умножить на 100, чтобы получить процентное соотношение.

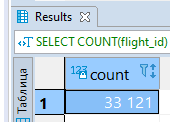
Понадобится подзапрос, вычисляющий общее количество перелетов:

**SELECT**

**COUNT**(flight\_id)

**FROM**

bookings.flights f2



SQL-запрос:

**SELECT**

f.aircraft\_code,

a.model,

**ROUND**(100.0 \* **COUNT**(f.flight\_id) / (**SELECT** **COUNT**(flight\_id) **FROM** bookings.flights f2), 2) **AS** aircraft\_percent

**FROM**

bookings.flights f

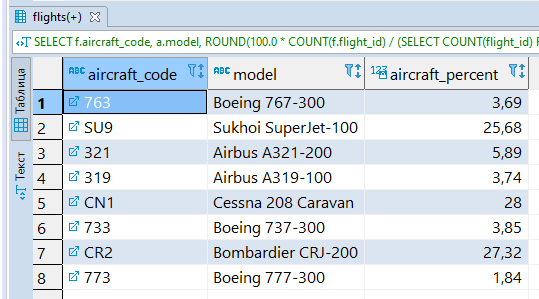
**LEFT** **JOIN** bookings.aircrafts a **ON**

f.aircraft\_code = a.aircraft\_code

**GROUP** **BY**

f.aircraft\_code,

a.model;



5.7 Были ли города, в которые можно добраться бизнес - классом дешевле, чем эконом-классом в рамках перелета?

Нужно воспользоваться CTE!

CTE играет роль представления, которое создается в рамках одного запроса и, не сохраняется как объект схемы. Имеет свой синтаксис, начинающийся с WITH…

У каждого полёта есть свой УНИКАЛЬНЫЙ идентификатор flight\_id с разным классом обслуживания fare\_conditions и разлётом цен (минимальной и максимальной стоимостью перелёта amount из таблицы ticket\_flights) в пределах одного полёта по классам обcлуживания.

Создадим общее табличное выражение (СTE), где в операторе HAVING задали, что имеем дело только с эконом- и бизнес-классом (всего у нас три класса обслуживания, третий отбрасываем, также нам важно, чтобы у нас во время перелёта были представлены и эконом- и бизнес-класс, фактически каждому идентификатору рейса соответствуют ровно два рассматриваемых класса обслуживания).

**WITH** prices **AS**

(

**SELECT**

f.flight\_id,

tf.fare\_conditions,

**MIN**(amount) min\_amount,

**MAX**(amount) max\_amount

**FROM** bookings.flights f

**RIGHT** **JOIN** bookings.ticket\_flights tf **ON** f.flight\_id = tf.flight\_id

**GROUP** **BY**

f.flight\_id,

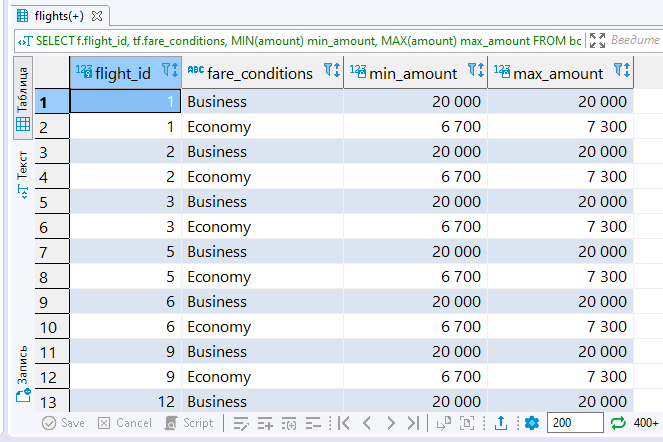
tf.fare\_conditions

**HAVING**

tf.fare\_conditions != 'Comfort' **AND** **COUNT**(tf.fare\_conditions) > 1

)

Выведем результаты запроса данного общего табличного выражения, то есть то, что находится в круглых скобках оператора WITH AS ():



Полный SQL-запрос с учётом вышепредставленного оператора:

**WITH** prices **AS**

(

**SELECT**

f.flight\_id,

tf.fare\_conditions,

**MIN**(amount) min\_amount,

**MAX**(amount) max\_amount

**FROM** bookings.flights f

**RIGHT** **JOIN** bookings.ticket\_flights tf **ON** f.flight\_id = tf.flight\_id

**GROUP** **BY**

f.flight\_id,

tf.fare\_conditions

**HAVING**

tf.fare\_conditions != 'Comfort' **AND** **COUNT**(tf.fare\_conditions) > 1

)

-- Финальный запрос, использующий CTE

**SELECT**

b.flight\_id,

b.min\_amount business\_min,

e.max\_amount economy\_max

**FROM** (**SELECT** \* **FROM** prices **WHERE** fare\_conditions = 'Business') b

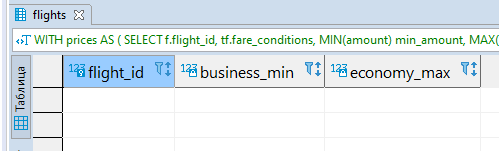
**LEFT** **JOIN** (**SELECT** \* **FROM** prices **WHERE** fare\_conditions = 'Economy') e

**ON** b.flight\_id = e.flight\_id

-- условие на сравнение цен в бизнес- и эконом- классе

**WHERE** b.min\_amount < e.max\_amount;

Стоит обратить на последнюю строчку данного запроса, здесь сравниваем стоимость перелёта на условие: ‘бизнес - классом дешевле, чем эконом-классом в рамках перелета?’.



Видно, что никаких перелётов с требуемыми условиями не осуществлялось. Если заменить знак < в операторе на знак =, также будет пустой вывод. А вот если поставить знак > вместо знака <, то естественно высветятся все варианты и можно смотреть города прибытия, соответствующие городу прибытия arrival\_city с помощью представления "bookings.flights\_v". Но поскольку мы ратуем за быстрое выполнение кода, то нет необходимости искать города прибытия.

По факту раз нет полётов, с нас не требуют названия городов прибытия, то городов, соответствующему нашему вопросы не было. Бизнес-классом лететь всегда дороже, чем эконом-классом.

5.8 Между какими городами нет прямых рейсов?

Следует найти всевозможные рейсы, прямые рейсы и вычесть полученные множества. Это и будет ответом.

Полный SQL-запрос:

-- в данное представление dep\_city перенесены всевозможные города   
--отправления из которых осуществляется вылет.

**CREATE** **OR** **REPLACE** **VIEW** bookings.dep\_city **AS** (

**SELECT** **DISTINCT** city dep\_city

**FROM** bookings.flights f

**LEFT** **JOIN** bookings.airports a **ON** f.departure\_airport = a.airport\_code

**ORDER** **BY** city);

-- в данное представление arr\_city перенесены всевозможные города  
-- прибытия, в которые осуществляется прилёт.

**CREATE** **OR** **REPLACE** **VIEW** bookings.arr\_city **AS** (

**SELECT** **DISTINCT** city arr\_city

**FROM** bookings.flights f

**LEFT** **JOIN** bookings.airports a

**ON** f.arrival\_airport = a.airport\_code

**ORDER** **BY** city);

-- все комбинации возможных перелётов

**SELECT** **CONCAT** (dep\_city, ' ', arr\_city) all\_possible\_flights

**FROM** (

**SELECT** \* **FROM** bookings.dep\_city dc, bookings.arr\_city ac

**WHERE** dep\_city != arr\_city) all\_possible;

-- существующие прямые перелёты

**SELECT** **CONCAT** (departure\_city, ' ', arrival\_city) existing\_flights

**FROM** (

**SELECT** **DISTINCT**

a1.city departure\_city,

a2.city arrival\_city

**FROM** bookings.flights f

**LEFT** **JOIN** bookings.airports a1

**ON** f.departure\_airport = a1.airport\_code

**LEFT** **JOIN** bookings.airports a2

**ON** f.arrival\_airport = a2.airport\_code

**ORDER** **BY** a1.city)

existing\_only;

**WITH** non\_existing\_direct **AS** (

-- выбираем всевозможные прямые перелёты

**SELECT** **CONCAT** (dep\_city, ' ', arr\_city) possible\_flights

**FROM** (

**SELECT** \* **FROM** bookings.dep\_city dc, bookings.arr\_city ac

**WHERE** dep\_city != arr\_city) all\_possible

**EXCEPT**

-- вычитаем существующие прямые перелёты

**SELECT** **CONCAT** (departure\_city, ' ', arrival\_city) existing\_flights

**FROM** (

**SELECT** **DISTINCT**

a1.city departure\_city,

a2.city arrival\_city

**FROM** bookings.flights f

**LEFT** **JOIN** bookings.airports a1

**ON** f.departure\_airport = a1.airport\_code

**LEFT** **JOIN** bookings.airports a2

**ON** f.arrival\_airport = a2.airport\_code

**ORDER** **BY** a1.city) existing\_only)

**SELECT** \* **FROM** non\_existing\_direct

**ORDER** **BY** possible\_flights;

Множество 1 Множество 2 Ответ

Всевозможные комбинации Реальные прямые рейсы Мн1\Мн2

5.9 Вычислите расстояние между аэропортами, связанными прямыми рейсами, сравните с допустимой максимальной дальностью перелетов в самолетах, обслуживающих эти рейсы В локальной базе координаты находятся в столбцах airports.longitude и airports.latitude.

Для решения поставленной задачи нужно взять прямые рейсы, вычислить по геоданным из таблиц расстояния между городами и сравнить с допустимой максимальной дальностью. Если происходит превышение, то подобные полёты не допустимо совершать.

Соответствующий SQL-запрос и логика выполнения:

**WITH** dist\_range\_comparison **AS** (

-- широта и долгота аэропорта

**WITH** ap\_lat\_lon **AS** (

**SELECT**

dep\_air,

dep\_name,

arr\_air,

arr\_name,

ac,

lat\_a,

lon\_a,

latitude lat\_b,

longitude lon\_b

**FROM**

(

**SELECT**

r.departure\_airport dep\_air,

r.departure\_airport\_name dep\_name,

r.arrival\_airport arr\_air,

r.arrival\_airport\_name arr\_name,

r.aircraft\_code ac,

a.latitude lat\_a,

a.longitude lon\_a

**FROM**

bookings.routes r

-- широта и долгота аэропорта отправления

**LEFT** **JOIN** bookings.airports a **ON**

r.departure\_airport = a.airport\_code) dep\_lat\_lon

-- широта и долгота аэропорта вылета и прилёта

**LEFT** **JOIN** bookings.airports a2 **ON**

dep\_lat\_lon.arr\_air = a2.airport\_code)

**SELECT**

dep\_air,

dep\_name,

arr\_air,

arr\_name,

model,

**ROUND**((**acos**(**sin**(**radians**(lat\_a))\***sin**(**radians**(lat\_b))+**cos**(**radians**(lat\_a))\***cos**(**radians**(lat\_b))\***cos**(**radians**(lon\_b - lon\_a))) \* 6371)::**numeric**, 1) dist\_km,

"range"

**FROM** ap\_lat\_lon

**LEFT** **JOIN** bookings.aircrafts ac

**ON** ap\_lat\_lon.ac = ac.aircraft\_code)

**SELECT**

\*,

(**CASE**

**WHEN** "range" - dist\_km > 100 **THEN** 'Допустимо'

**WHEN** "range" - dist\_km <= 100 **AND** "range" - dist\_km > 0 **THEN** 'На пределе'

**WHEN** "range" - dist\_km <= 0 **THEN** 'Не допустимо'

**END**) flight\_admissibility

**FROM** dist\_range\_comparison

-- WHERE "range" - dist\_km <= 0

**ORDER** **BY** dist\_km **DESC**;



Кратчайшее расстояние между двумя точками A и B на земной поверхности (если принять ее за сферу) определяется зависимостью:

d = arccos {sin(latitude\_a)·sin(latitude\_b) + cos(latitude\_a)·cos(latitude\_b)·cos(longitude\_a - longitude\_b)}, где latitude\_a и latitude\_b — широты, longitude\_a, longitude\_b — долготы данных пунктов, d — расстояние между пунктами измеряется в радианах длиной дуги большого круга земного шара.

Расстояние между пунктами, измеряемое в километрах, определяется по формуле:

L = d·R, где R = 6371 км — средний радиус земного шара.