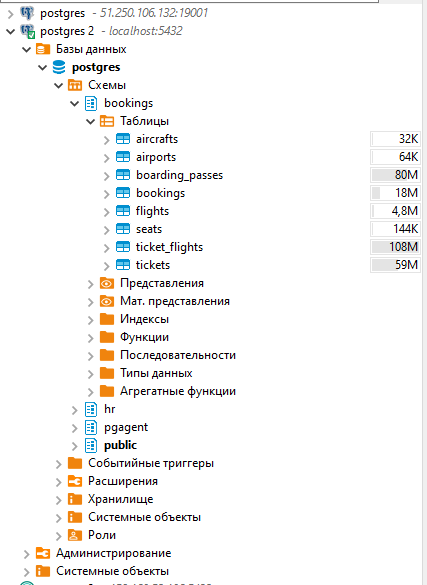
**Проектная работа по модулю**

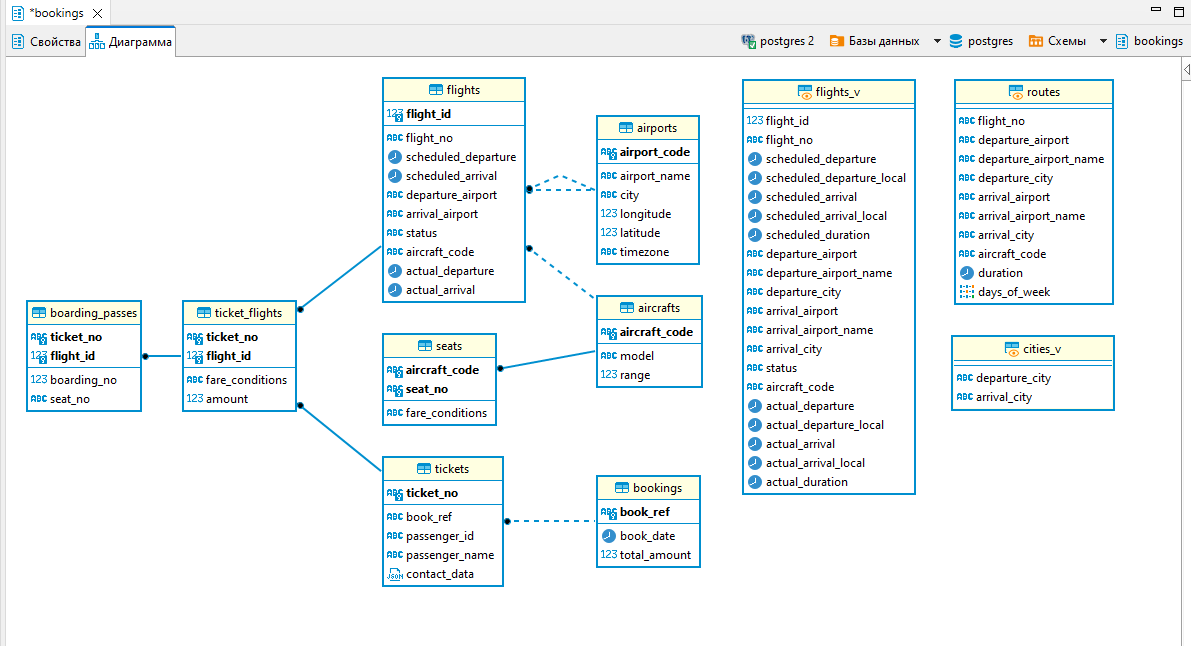
**«SQL и получение данных»**

(Группа DA-CP-2 Емельянова Лия)

1. В работе использовался локальный тип подключения (восстановление из \*.backup файла).



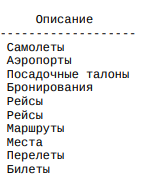
1. Скриншот ER-диаграммы из D'Beaver.



1. Краткое описание БД.

Основной сущностью базы данных является бронирование (bookings).

База данных состоит из 8 таблиц (aircrafts, airports, boarding\_passes, bookings, flights, seats, ticket\_flights, tickets), представления (flights\_v) и материализованного представления (routes).



4. Развернутый анализ БД - описание таблиц, логики, связей и бизнес области.

Бизнес область базы данных - авиаперевозки по России.

Бронирование (bookings) является основной сущностью базы данных. Бронь (book\_ref) может быть выписана как на одного, так и на нескольких пассажиров. В билете может быть несколько перелетов (flight\_id), если между аэропортами маршрута нет прямых рейсов, либо билет «туда/обратно».

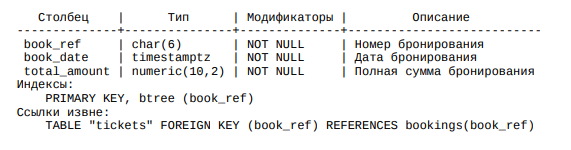
Вся информация о перелетах содержится в таблице flights (номер рейса, аэропорт вылета/прилета, время вылета/прилета). Данные по аэропортам и городам их нахождения - таблица airports.

При регистрации на рейс пассажиру выдается посадочный талон (boarding\_no), в котором указывается место в самолете (seat\_no). Пассажир может зарегистрироваться только на тот рейс, который есть у него в билете.

Данные о самолетах, количестве мест и их классах обслуживания содержатся в таблицах aircrafts и seats.

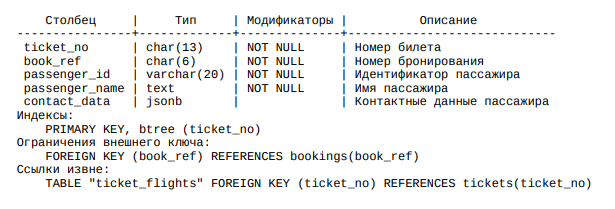
4.1 (**bookings**). Пассажир заранее (book\_date, максимум за месяц до рейса) бронирует (bookings) билет себе и, возможно, нескольким другим пассажирам.

Бронирование идентифицируется номером (book\_ref, шестизначная комбинация букв и цифр). Поле total\_amount хранит общую стоимость включенных в бронирование перелетов всех пассажиров.



4.2 (**tickets**). В одно бронирование можно включить несколько пассажиров, каждому из которых оформляется отдельный билет (tickets). Билет имеет

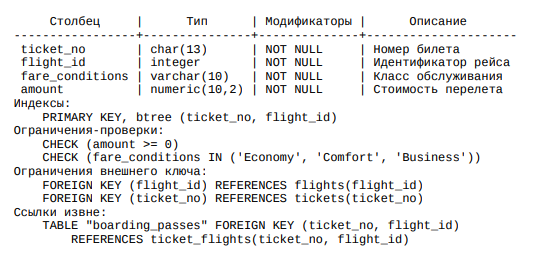
уникальный номер (ticket\_no), состоящий из 13 цифр, содержит идентификатор пассажира (passenger\_id) — номер документа, удостоверяющего личность, — его фамилию и имя (passenger\_name) и контактную информацию (contact\_date).



4.3 (**ticket\_flights**). Билет включает один или несколько перелетов (ticket\_flights). В схеме данных нет жесткого ограничения, но предполагается,

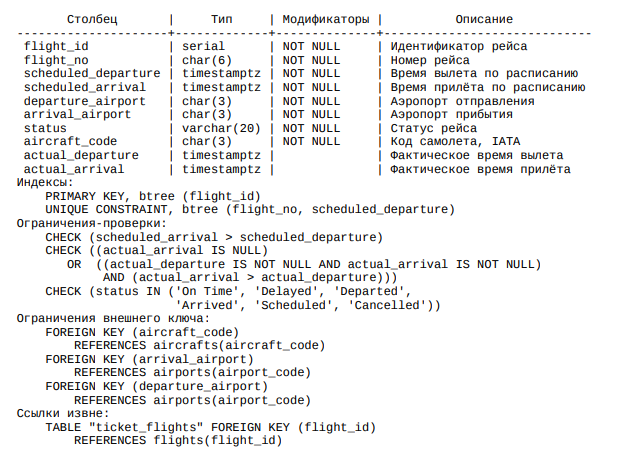
что все билеты в одном бронировании имеют одинаковый набор перелетов. Перелет соединяет билет с рейсом и идентифицируется их номерами.

Для каждого перелета указываются его стоимость (amount) и класс обслуживания (fare\_conditions).

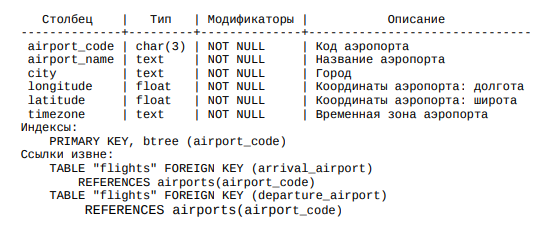


4.4 (**flights**). Каждый рейс (flights) следует из одного аэропорта (airports) в другой. Рейсы с одним номером имеют одинаковые пункты вылета и назначения, но будут отличаться датой отправления. Рейс всегда соединяет две точки - аэропорты вылета (departure\_airport) и прибытия (arrival\_airport).

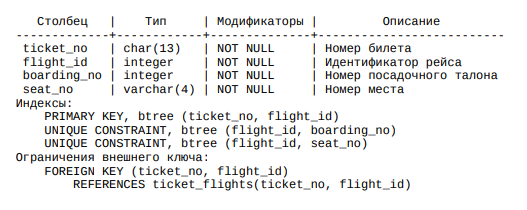
У каждого рейса есть запланированные дата и время вылета (scheduled\_departure) и прибытия (scheduled\_arrival). Реальные время вылета (actual\_departure) и прибытия (actual\_arrival) могут отличаться. Статус рейса (status) может принимать одно из следующих значений: Scheduled, On Time, Delayed, Departed, Arrived, Cancelled.



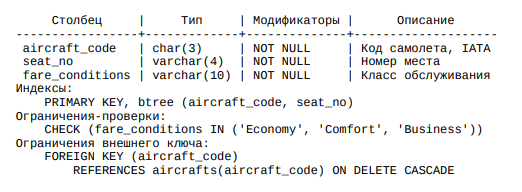
4.5 (**airports**). Аэропорт идентифицируется трехбуквенным кодом (airport\_code) и имеет свое имя (airport\_name). Для города не предусмотрено отдельной сущности, но название (city) указывается и может служить для того, чтобы определить аэропорты одного города. Также указывается широта (longitude), долгота (latitude) и часовой пояс (timezone).



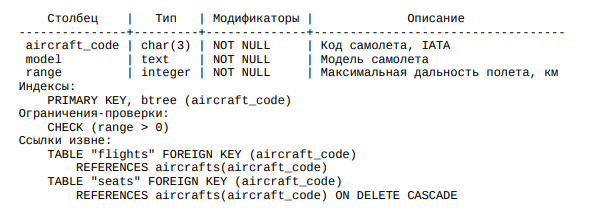
4.6 (**boarding\_passes**). При регистрации на рейс пассажиру выдается посадочный талон. Посадочным талонам присваиваются последовательные номера (boarding\_no) в порядке регистрации пассажиров на рейс (этот номер будет уникальным только в пределах данного рейса). В посадочном талоне указывается номер места (seat\_no).



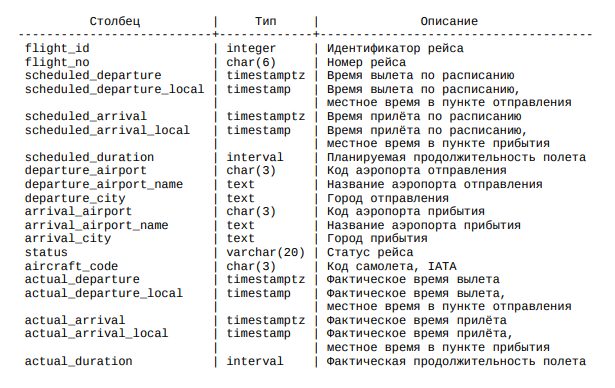
4.7 (**seats**). Количество мест (seats) в самолете и их распределение по классам обслуживания зависит от модели самолета (aircrafts), выполняющего рейс. Места определяют схему салона каждой модели. Каждое место определяется своим номером (seat\_no) и имеет закрепленный за ним класс обслуживания (fare\_conditions) — Economy, Comfort или Business.



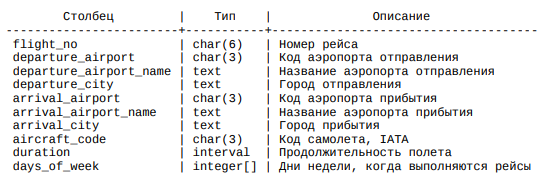
4.8 (**aircrafts**). Каждая модель воздушного судна идентифицируется своим трехзначным кодом (aircraft\_code). Указывается также название модели (model) и максимальная дальность полета в километрах (range).



4.9 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 полета (scheduled\_duration, actual\_duration).



4.10 Таблица рейсов содержит избыточность: из нее 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**.



Анализ информации БД помогает решить следующие бизнес-задачи:

1. Анализ аэропортов - какие более нагружены, чтобы оптимизировать поток пассажиров.

2. Какая часть билетов только бронируется, но не выкупается и, по каким направлениям.

3. На каких рейсах процент мест в самолете остается свободным, чтобы провести оптимизацию, и сократить количество самолётов, летающих «впустую» (например, снижение пассажиропотока на таких направлениях).

4. Выяснить какие маршруты самые популярные.

5. Планирование закупки новых самолетов, исходя из предполагаемых маршрутов их использования.

6. Продумать накопительную систему скидок для постоянных клиентов.

5. Список SQL запросов с описанием логики их выполнения:

= УСТАНАВЛИВАЕМ СОЕДИНЕНИЕ И ВЫБРАЕМ СХЕМУ BOOKINGS =

**SET** search\_path = bookings;

1) *В каких городах больше одного аэропорта?* – Москва и Ульяновск

Для решения был использован запрос к таблице airports с применением агрегатной функции и группировки по столбцу city, далее сортировка.

Вариант решения 1:

**select** city **from** airports

**group** **by** city

**having** **count**(city) > 1

**order** **by** city;

Вариант решения 2:

**select** a.airport\_code **as** code,

a.airport\_name,

a.city,

a.longitude,

a.latitude,

a.timezone

**from** airports a

**where** a.city **in** (

**select** aa.city

**from** airports aa

**group** **by** aa.city

**having** **count**(\*) > 1

)

**order** **by** a.city, a.airport\_code;

2) *В каких аэропортах есть рейсы, выполняемые самолетом с максимальной дальностью перелета?* – Внуково, Домодедово, Кольцово, Пермь, Сочи, Толмачёво, Шереметьево.

С помощью подзапроса находим самолёт с самой максимальной дальностью перелёта применяя ограничение LIMIT, затем с помощью оператора UNION объединяем таблицы flights\_v и aircrafts по коду самолёта анализируя аэропорта прибытия и отправления.

**select** departure\_airport, departure\_airport\_name

**from** flights\_v fv

**where** fv.aircraft\_code = (**select** a.aircraft\_code **from** aircrafts a **order** **by** a."range" **desc** **limit** 1)

**union**

**select** arrival\_airport, arrival\_airport\_name

**from** flights\_v fv

**where** fv.aircraft\_code = (**select** a.aircraft\_code **from** aircrafts a **order** **by** a."range" **desc** **limit** 1);

3) *Вывести 10 рейсов с максимальным временем задержки вылета.*

В таблице flights вычисляем для каждого рейса время задержки вылета (от фактического отнимаем время вылета по расписанию) указывая обязательное условие is not null. Далее сортируем по убыванию информацию по задержке рейса и ограничиваем вывод до 10.

**select**

actual\_departure - scheduled\_departure **as** departure\_delay,

flight\_no,

departure\_airport

**from** flights

**where** actual\_departure - scheduled\_departure **is** **not** **null**

**order** **by** departure\_delay **desc**

**limit** 10;

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

Чтобы получить информацию по бронированию с нулевыми значениями к основной таблице bookings присоединяем с помощью left join таблицы tickets и boarding\_passes. Далее пишем условие что номер посадочного талона должен быть is null.

**select** b.book\_ref, bp.boarding\_no, bp.ticket\_no

**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**;

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.*

С помощью СТЕ находим общее количество мест в самолёте. Далее из общего количества мест вычитаем посадочные талоны на этом рейсе тем самым получаем свободные места. Чтобы получить процентное отношение к общему количеству - мы свободные места делим на общее количество мест и умножаем на 100. Применяем оконную функцию для подсчёта вылетевших за день из аэропорта людей.

**with** cte **as** (

**select** s.aircraft\_code, **count**(s.seat\_no) **as** "count", a.model

**from** seats s

**join** aircrafts a **on** a.aircraft\_code = s.aircraft\_code

**group** **by** s.aircraft\_code, a.model

)

**select**

departure\_airport,

actual\_departure ,

cte."count" - **count**(bp.seat\_no) **as** available\_seats,

(((cte."count" - **count**(bp.seat\_no))::**numeric** / cte."count")::**numeric**(32,2)) \* 100 **as** percentage\_of\_available\_seats,

**sum**(**count**(bp.seat\_no)) **over** (**partition** **by** f.actual\_departure::**date**, f.departure\_airport **order** **by** f.actual\_departure) **as** cumulative\_total

**from** boarding\_passes bp

**join** flights f **on** f.flight\_id = bp.flight\_id

**join** cte **on** cte.aircraft\_code = f.aircraft\_code

**group** **by** f.flight\_id, cte."count";

6) *Найдите процентное соотношение перелетов по типам самолетов от общего количества.*

С помощью подзапроса считаем количество перелётов для каждого самолёта от их общего количества. Далее, используя значения, полученные в подзапросе, делим на оконную функцию для подсчёта общего количества перелётов и умножаем на 100. Таким образом находим процент от перелёта в рамках функции round.

**select** model, (**round**("count"::**numeric** / (**sum**("count") **over** ()), 2) \* 100) **as** percentage\_of\_total\_number\_of\_flights

**from**(

**select** **count**(flight\_id) **as** "count", model

**from** flights f

**join** aircrafts a **on** a.aircraft\_code = f.aircraft\_code

**group** **by** model

) f;

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

В первом СТЕ\_1 выводим цену эконома в рамках перелёта, в СТЕ\_2 выводим стоимость билета бизнес - класса в рамках перелёта. Агрегация нужна для нахождения максимальной цены билета для эконома и минимальной цены бизнес - класса в рамках перелёта. Далее пишем условие сравнения двух СТЕ и если условие выполнится, то выводим город.

**with** cte\_1 **as** (

**select** **distinct** tf.flight\_id, **max**(amount) **as** price\_econom, fare\_conditions

**from** ticket\_flights tf

**where** tf.fare\_conditions = 'Economy'

**group** **by** flight\_id, fare\_conditions

**order** **by** flight\_id

) ,

cte\_2 **as** (

**select** **distinct** tf.flight\_id, **min**(amount) **as** price\_business , fare\_conditions

**from** ticket\_flights tf

**where** tf.fare\_conditions = 'Business'

**group** **by** flight\_id, fare\_conditions

**order** **by** flight\_id

)

**select** a.city **as** city\_of\_arrival

**from** cte\_1

**join** cte\_2 **on** cte\_2.flight\_id = cte\_1.flight\_id

**join** flights f **on** f.flight\_id = cte\_1.flight\_id

**join** airports a **on** a.airport\_code = f.arrival\_airport

**where** price\_business < price\_econom;

8) *Между какими городами нет прямых рейсов?*

С помощью декартово произведения (cross join) находим связи город – город. Используем оператор distinct чтобы убрать дубли Москвы и Ульяновска(так как там несколько аэропортов) и оператор where для того чтобы убрать связки городов на себя. Из представления flights\_v выбираем города с рейсами и оператором except выводим конечный результат, указанный в задании (Точнее делаем представление (дубль flights\_v с интересующими нас значениями)).

**create** **view** cities\_v **as**

**select** v.departure\_city, v.arrival\_city

**from** flights\_v v

**select** **distinct** a.city, a1.city

**from** airports a

**cross** **join** airports a1

**where** a.city != a1.city

**except**

**select** c.departure\_city, c.arrival\_city

**from** cities\_v c;

9) *Вычислите расстояние между аэропортами, связанными прямыми рейсами, сравните с допустимой максимальной дальностью перелетов в самолетах, обслуживающих эти рейсы.*

С помощью СТЕ находим дистанцию между аэропортами - для этого берём данные из таблицы airports и связываем их с таблицей flight (аэропортом отправления и аэропортом прибытия (departure\_airport и arrival\_airport соответственно)). Далее вводим формулу, указанную в задании, и получаем расстояние между ними. Затем берём значения максимальной дальности полёта из таблицы aircrafts и проверяем «если максимальная дальность полёта больше чем дистанция маршрута то true если нет - false». Так же был использован оператор distinct так как надо отсечь дублирующие значения.

**with** cte **as** (

**select** f.departure\_airport, dep.longitude, dep.latitude, f.arrival\_airport, arr.longitude, arr.latitude,

f.arrival\_airport, f.aircraft\_code,

**round**(((**acos**((**sind**(dep.latitude)\***sind**(arr.latitude) + **cosd**(dep.latitude) \* **cosd**(arr.latitude) \* **cosd**((dep.longitude - arr.longitude))))) \* 6371)::**numeric**, 2)

**as** distance\_airports ,

f.flight\_no,

dep.airport\_name **as** departure\_airport\_name,

arr.airport\_name **as** arrival\_airport\_name

**from**

flights f,

airports dep,

airports arr

**where** f.departure\_airport = dep.airport\_code **and** f.arrival\_airport = arr.airport\_code

)

**select** **distinct** cte.departure\_airport\_name, cte.arrival\_airport\_name, cte.distance\_airports,

a.**range** **as** aircraft\_flight\_distance,

**case**

**when** **range** > distance\_airports

**then** 'Полёт прошёл удачно'

**else** 'КРУШЕНИЕ САМОЛЁТА'

**end** **result**

**from** aircrafts a

**join** cte **on** cte.aircraft\_code = a.aircraft\_code