

# Итоговая работа SQL-7-1 по модулю “SQL и получение данных”

---

Студент курса data science: Григорьев В.С.

# Содержание

1. [Введение.](#)
2. [ER – диаграмма облачной базы данных.](#)
3. [Описание облачной базы данных, ее таблиц и представлений.](#)
4. [SQL – запросы и их описание, применение функций при решении практических заданий.](#)
5. [Самостоятельно развернутая база данных на локальном компьютере.](#)
6. [ER – диаграмма локальной базы данных.](#)

# 1. Введение.

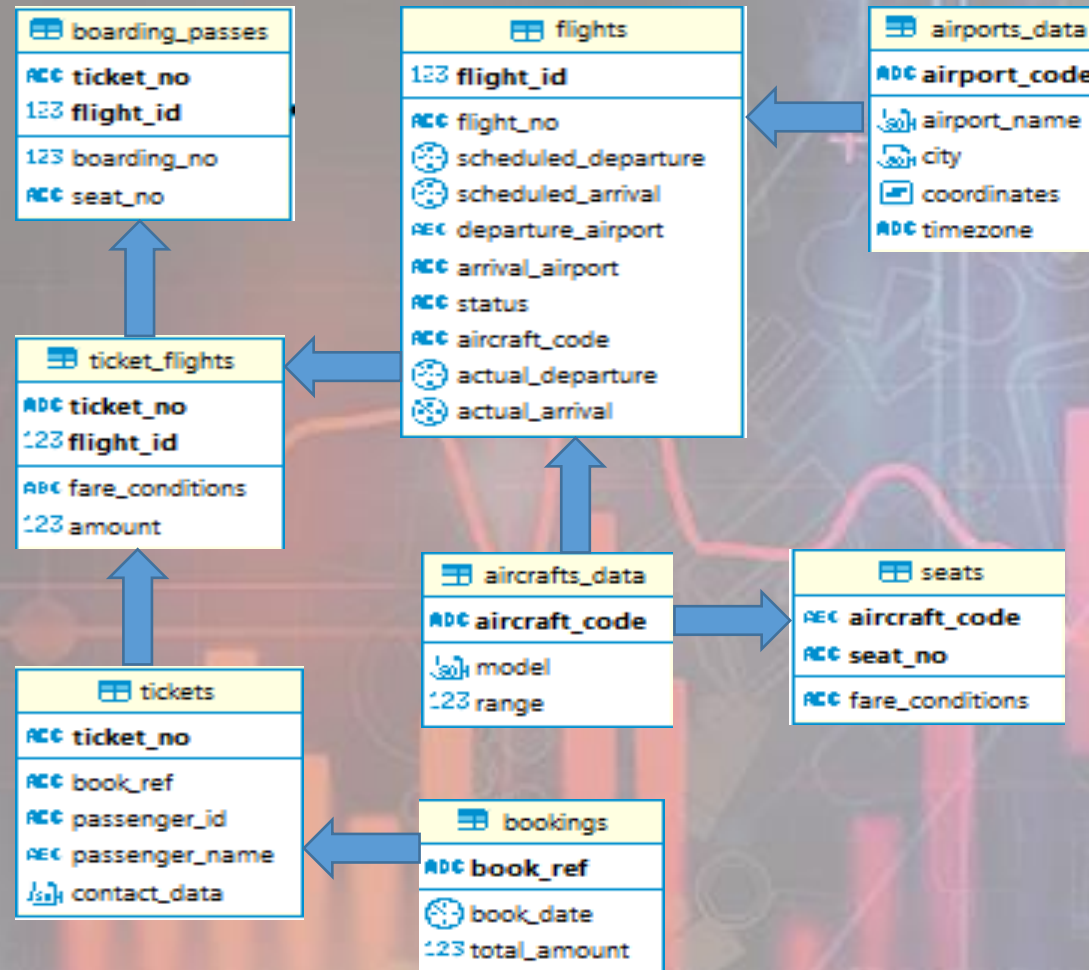
Целью данного курсового проекта является закрепление пройденных занятий и полученных знаний по курсу **SQL-7-1**.

Задачи:

- В каких городах больше одного аэропорта?
- В каких аэропортах есть рейсы, которые обслуживаются самолетами с максимальной дальностью перелетов?
- Были ли брони, по которым не совершались перелеты?
- Самолеты каких моделей совершают наибольший % перелетов?
- Были ли города, в которые можно добраться бизнес - классом дешевле, чем эконом-классом?
- Узнать максимальное время задержки вылетов самолетов.



## 2. ER – диаграмма облачной базы данных




### 3. Описание облачной базы данных, ее таблиц и представлений.

Данная схема данных, состоит из восьми таблиц и нескольких представлений. В качестве предметной области выбраны авиаперевозки по России.

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 перелетов (ticket\_flights). Несколько перелетов 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 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.

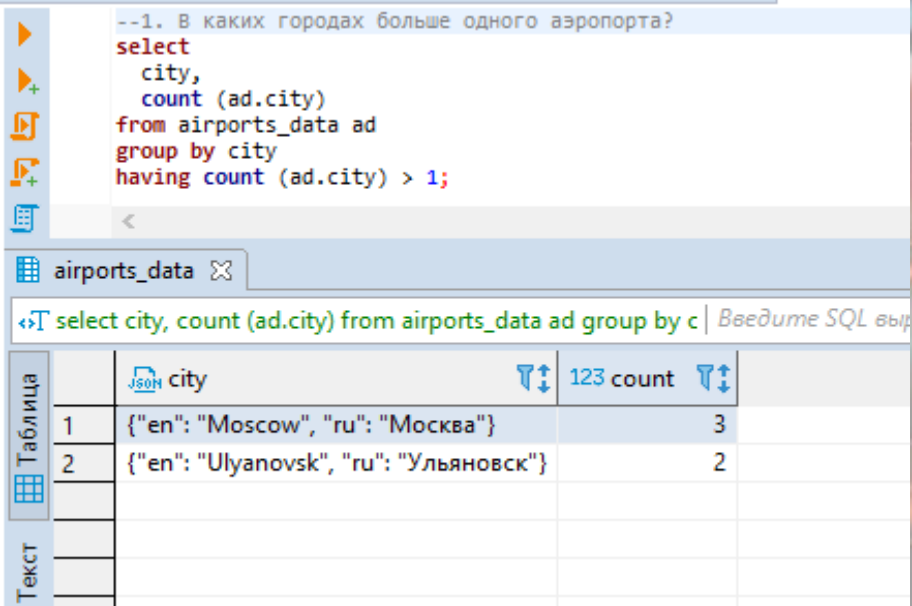
При регистрации на рейс пассажиру выдается посадочный талон (boarding\_passes), в котором указано место в самолете. Пассажир может зарегистрироваться только на тот рейс, который есть у него в билете. Комбинация рейса и места в самолете должна быть уникальной, чтобы не допустить выдачу двух посадочных 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 самолета имеет только одну компоновку салона. Схема данных не контролирует, что места в посадочных талонах соответствуют имеющимся в самолете (такая проверка может быть сделана с использованием табличных триггеров или в приложении).



## 4. SQL – запросы и их описание, применение функций при решении практических заданий.

- В каких городах больше одного аэропорта?

```
select  
    city,  
    count (ad.city)  
from airports_data ad  
group by city  
having count (ad.city) > 1;
```



The screenshot shows a SQL query editor with the following query:

```
--1. В каких городах больше одного аэропорта?  
select  
    city,  
    count (ad.city)  
from airports_data ad  
group by city  
having count (ad.city) > 1;
```

Below the query, the results are displayed in a table. The table has two columns: 'city' and 'count'. The results are as follows:

	city	count
1	{"en": "Moscow", "ru": "Москва"}	3
2	{"en": "Ulyanovsk", "ru": "Ульяновск"}	2

Для выполнения данного запроса выбираем таблицу «airports\_data» присваиваем алиас «ad», группируем по городу и с помощью оператора «having» делаем условие фильтрации для подсчитанного столбца «count», устанавливая значение больше 1

- В каких аэропортах есть рейсы, которые обслуживаются самолетами с максимальной дальностью перелетов?

```
with cte_vlad as (  
  select  
    f.aircraft_code as air_code,  
    flight_id,  
    flight_no,  
    departure_airport,  
    arrival_airport,  
    model  
  from flights f  
  inner join aircrafts_data af on f.aircraft_code = af.aircraft_code)  
select * from cte_vlad  
where air_code = '773'; -- взят из таблицы "aircraft_data"
```



ия данного запроса создаем табличное выражение (CTE) выбираем из «t» необходимые нам столбцы, с помощью оператора «inner join» с «aircraft\_data». Выводим необходимые нам столбцы из CTE с условием «where» air\_code = '773'. Выводим результат с данным кодом, из таблицы aircraft\_data, имеет наибольшую дальность полета.

```
--2. В каких аэропортах есть рейсы, которые обслуживаются самолетами с максимальной дальностью перелетов?
with cte_vlad as (
    select
        f.aircraft_code as air_code,
        flight_id,
        flight_no,
        departure_airport,
        arrival_airport,
        model
    from flights f
    inner join aircrafts_data af on f.aircraft_code = af.aircraft_code)
select * from cte_vlad
where air_code = '773'; -- взят из таблицы "aircraft_data"
```

air_code	flight_id	flight_no	departure_airport	arrival_airport	model
773	10,455	PG0277	SVO	OVB	{ "en": "Boeing 777-300", "ru": "Боинг 777-300" }
773	485	PG0222	DME	OVB	{ "en": "Boeing 777-300", "ru": "Боинг 777-300" }
773	486	PG0222	DME	OVB	{ "en": "Boeing 777-300", "ru": "Боинг 777-300" }
773	487	PG0222	DME	OVB	{ "en": "Boeing 777-300", "ru": "Боинг 777-300" }
773	488	PG0222	DME	OVB	{ "en": "Boeing 777-300", "ru": "Боинг 777-300" }
773	489	PG0222	DME	OVB	{ "en": "Boeing 777-300", "ru": "Боинг 777-300" }
773	490	PG0222	DME	OVB	{ "en": "Boeing 777-300", "ru": "Боинг 777-300" }
773	491	PG0222	DME	OVB	{ "en": "Boeing 777-300", "ru": "Боинг 777-300" }
773	492	PG0222	DME	OVB	{ "en": "Boeing 777-300", "ru": "Боинг 777-300" }

- Были ли брони, по которым не совершались перелеты?

**select**

t.book\_ref,

bp.ticket\_no as "boarding\_ticket",

t.passenger\_name

**from boarding\_passes bp**

**full join** ticket\_flights tf **on** bp.ticket\_no = tf.ticket\_no

**full join** tickets t **on** t.ticket\_no = tf.ticket\_no

**full join** bookings b **on** b.book\_ref = t.book\_ref

**where bp.ticket\_no is null**

Для выполнения данного запроса выбираем из таблицы «boarding\_passes» и «ticket» необходимые нам столбцы, соединяем с помощью оператора «full join» с таблицами «ticket\_flight», «ticket», «bookings». Далее при помощи команды «where» задаем условие где алиас «boarding\_ticket» is null.

```
--3.Были ли брони, по которым не совершались перелеты?
select
  t.book_ref,
  bp.ticket_no as "boarding_ticket",
  t.passenger_name
from boarding_passes bp
full join ticket_flights tf on bp.ticket_no = tf.ticket_no
full join tickets t on t.ticket_no = tf.ticket_no
full join bookings b on b.book_ref = t.book_ref
where bp.ticket_no is null
```

tickets(+) X

SQL select t.book\_ref, bp.ticket\_no as "boarding\_ticket", t.passenger\_name | Введите SQL выражение

		ABC book_ref	ABC boarding_ticket	ABC passenger_name
Таблица	1	82A3EB	[NULL]	TATYANA STEPANOVA
	2	9DE56E	[NULL]	VERONIKA ZHUKOVA
	3	B5DEDB	[NULL]	VALERIY YAKOVLEV
	4	399375	[NULL]	NATALYA KISELEVA
Текст	5	F5ACF1	[NULL]	EKATERINA POPOVA
	6	F03ACE	[NULL]	ANDREY ABRAMOV
	7	2BF7EA	[NULL]	ELENA KUZNECOVA



- Самолеты каких моделей совершают наибольший % перелетов?

**select**

af.model,

**count** (flight\_id) \* 100 / (**select count** (\*) from flights) as percent

**from** flights f

**join** aircrafts\_data af **ON** f.aircraft\_code = af.aircraft\_code

**group by** model

**order by** "percent" desc



ия данного запроса выбираем из таблицы  
 помощью оператора «join» с таблицей «air  
 порирования по «model» с условием порядка

```
--4.Самолеты каких моделей совершают наибольший % перелетов?

select
  af.model,
  count (flight_id) * 100 / (select count (*) from flights) as percent
from flights f
  join aircrafts_data af ON f.aircraft_code = af.aircraft_code
group by model
order by "percent" desc
```

aircrafts\_data

```
select af.model, count (flight_id) * 100 / (select count (*) fr
```

model	percent
1 {"en": "Cessna 208 Caravan", "ru": "Сессна 208 Караван"}	28
2 {"en": "Bombardier CRJ-200", "ru": "Бомбардье CRJ-200"}	27
3 {"en": "Sukhoi Superjet-100", "ru": "Сухой Суперджет-100"}	25
4 {"en": "Airbus A321-200", "ru": "Аэробус A321-200"}	5
5 {"en": "Boeing 737-300", "ru": "Боинг 737-300"}	3
6 {"en": "Airbus A319-100", "ru": "Аэробус A319-100"}	3
7 {"en": "Boeing 767-300", "ru": "Боинг 767-300"}	3
8 {"en": "Boeing 777-300", "ru": "Боинг 777-300"}	1

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

```
with t1 as ( select f.departure_city || '-' ||  
f.arrival_city as city,  
tf.amount as "amount_EC"--эконом класс  
from ticket_flights tf  
join flights_v f ON tf.flight_id = f.flight_id  
where tf.fare_conditions = 'Economy'  
group by tf.amount, f.departure_city,  
f.arrival_city),  
t2 as (select f.departure_city || '-' ||  
f.arrival_city as city,  
tf.amount as "amount_BC"--бизнес класс  
from ticket_flights tf  
join flights_v f ON tf.flight_id = f.flight_id
```

```
where tf.fare_conditions = 'Business'  
group by tf.amount, f.departure_city, f.arrival_city)  
select  
t1.city,  
(select "amount_EC" - "amount_BC") as total  
from t1  
join t2 ON t1.city = t2.city  
order by "total" desc
```



Для выполнения данного запроса создаем два обобщенных табличных выражения (СТЕ) выбираем из таблицы «ticket\_flight» необходимые нам столбцы, соединяем с помощью оператора «join» с таблицей «flights\_v». При помощи команды «where» задаем `tf.fare_conditions = 'Economy'`. Аналогично создаем еще одну таблицу где `tf.fare_conditions = 'Business'`. Делаем запрос с указанием необходимых столбцов и объединением их по городам, где при помощи вычитания из дешевых билетов - дорогих, получаем отрицательные значения. Это говорит о том, что за весь период стоимость дешевых билетов не превышала стоимости дорогих.

Если бы у нас было превышение по указанным направлениям – это отразилось бы в значении большем нуля.

В данном случае ответ - отрицательный. Нет городов куда можно добраться бизнесом дешевле чем экономом

```
--5.Были ли города, в которые можно добраться бизнес - классом дешевле, чем эконом-классом?  
|  
with t1 as  
(  
  select  
    f.departure_city || '-' || f.arrival_city as city,  
    tf.amount as "amount_EC"--эконом класс  
  from ticket_flights tf  
    join flights_v f ON tf.flight_id = f.flight_id  
  where tf.fare_conditions = 'Economy'  
  group by tf.amount, f.departure_city, f.arrival_city  
)  
t2 as  
(  
  select  
    f.departure_city || '-' || f.arrival_city as city,  
    tf.amount as "amount_BC"--бизнес класс  
  from ticket_flights tf  
    join flights_v f ON tf.flight_id = f.flight_id  
  where tf.fare_conditions = 'Business'  
  group by tf.amount, f.departure_city, f.arrival_city  
)  
select  
  t1.city,  
  (select "amount_EC" - "amount_BC") as total  
from t1  
  join t2 on t1.city = t2.city  
order by "total" desc
```

Result

with t1 as ( select f.departure\_city || '-' || f.arrival\_city as city, Введите SQL выражение чтобы отфильтровать

Таблица	city	total
1	Брянск-Москва	-5,800
2	Москва-Брянск	-5,800
3	Новокузнецк-Новосибирск	-5,800
4	Новосибирск-Новокузнецк	-5,800
5	Пермь-Екатеринбург	-6,000
6	Екатеринбург-Пермь	-6,000

- Узнать максимальное время задержки вылетов самолетов.

**select**

scheduled\_departure,

actual\_departure,

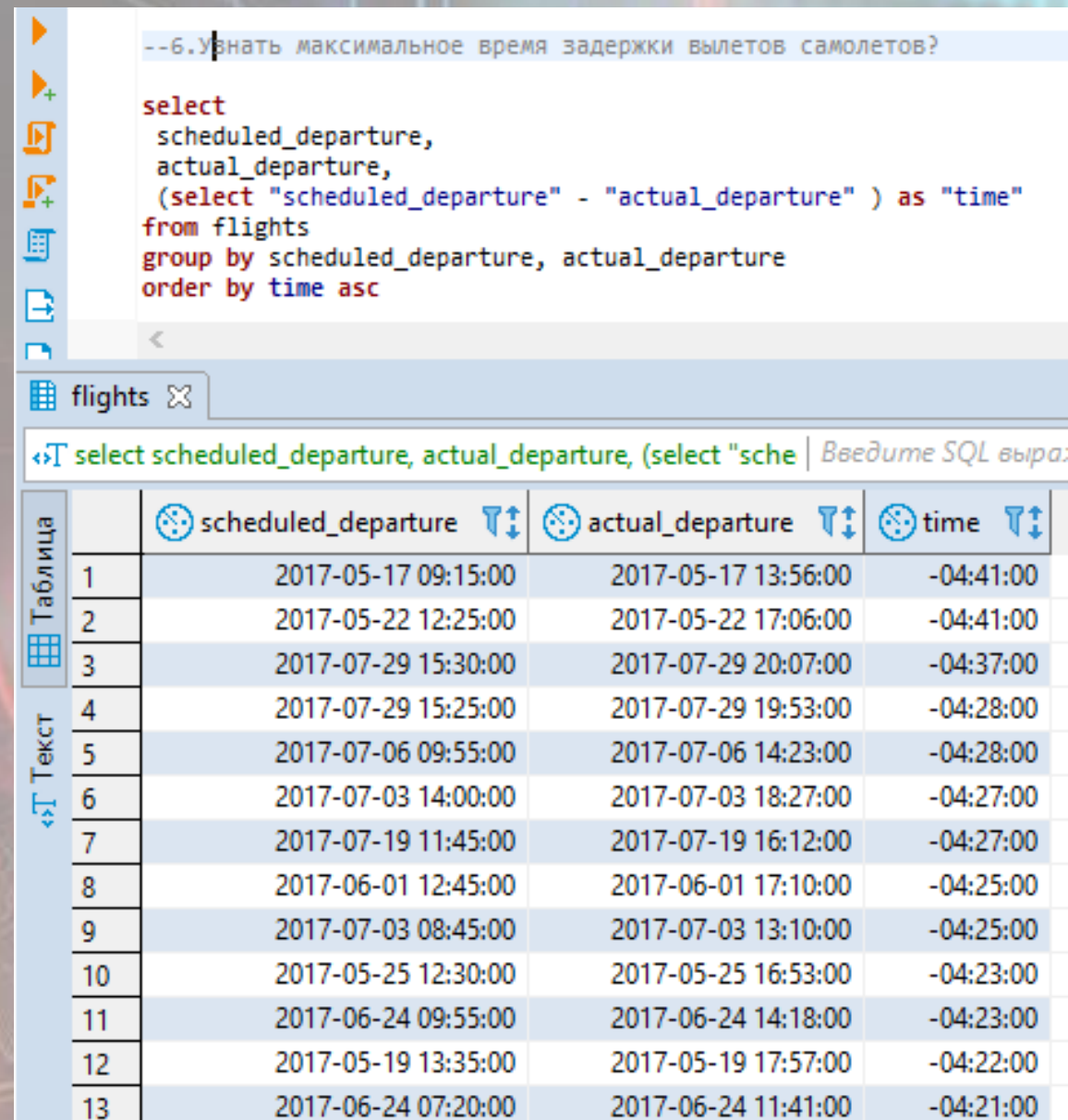
**(select** "scheduled\_departure" -  
"actual\_departure" ) **as** "time"

**from** flights

**group by** scheduled\_departure,  
actual\_departure

**order by** time **asc**

Для выполнения данного запроса выбираем из таблицы «flight» необходимые нам столбцы. Применяем подзапрос в котором указываем необходимые нам действия, а именно - вычитание из столбца «scheduled\_departure» столбец «actual\_departure» тем самым получаем задержку рейсов относительно запланированного вылета. Группируя «group by» и задавая порядок «order by» по возрастанию, получаем значения в столбце «time»



```
--6. Узнать максимальное время задержки вылетов самолетов?  
  
select  
    scheduled_departure,  
    actual_departure,  
    (select "scheduled_departure" - "actual_departure" ) as "time"  
from flights  
group by scheduled_departure, actual_departure  
order by time asc
```

	scheduled_departure	actual_departure	time
1	2017-05-17 09:15:00	2017-05-17 13:56:00	-04:41:00
2	2017-05-22 12:25:00	2017-05-22 17:06:00	-04:41:00
3	2017-07-29 15:30:00	2017-07-29 20:07:00	-04:37:00
4	2017-07-29 15:25:00	2017-07-29 19:53:00	-04:28:00
5	2017-07-06 09:55:00	2017-07-06 14:23:00	-04:28:00
6	2017-07-03 14:00:00	2017-07-03 18:27:00	-04:27:00
7	2017-07-19 11:45:00	2017-07-19 16:12:00	-04:27:00
8	2017-06-01 12:45:00	2017-06-01 17:10:00	-04:25:00
9	2017-07-03 08:45:00	2017-07-03 13:10:00	-04:25:00
10	2017-05-25 12:30:00	2017-05-25 16:53:00	-04:23:00
11	2017-06-24 09:55:00	2017-06-24 14:18:00	-04:23:00
12	2017-05-19 13:35:00	2017-05-19 17:57:00	-04:22:00
13	2017-06-24 07:20:00	2017-06-24 11:41:00	-04:21:00



## 5. Самостоятельно развернутая база данных на локальном компьютере.

Для развертывания СУБД на локальном компьютере необходимо запастись терпением, стальными нервами, а так же иметь при себе сам бэкап базы, шоколадку и кофе. Иначе ничего не получится.

Скачиваем сам PostgreSQL с [официального сайта](#) в соответствии с вашей системой.

Устанавливаем полный пакет с pgAdmin4 и начинаем.

Закончив установку необходимо убедиться что наш «pgAdmin4» не требует пароля, поскольку этот пункт при установке отсутствует, но появляется запрос при запуске «pgAdmin4». Это дело затираем в файле C:\Program Files\PostgreSQL\11\data\pg\_hba.conf

Ищем такие строчки

```
# IPv4 local connections:  
host all all 127.0.0.1/32 md5  
host all all 0.0.0.0/0 md5  
# IPv6 local connections:  
host all all ::1/128 md5
```

Меняем md5 на trust. Удаляем файл pgpass.conf – если есть.

После чего «pgAdmin4» дает нам возможность создать свою базу. Далее через командную строку подгружаем бекап нашей базы.

**ВАЖНО.** Папка где хранится наша база должна быть на латинице!!!

Win+r - > cmd ->Enter

1. C:\Users\Vlad>cd "C:\\Program Files\PostgreSQL\11\bin\" – путь к PostgreSQL
2. C:\Program Files\PostgreSQL\11\bin>psql.exe -U postgres demo < c:\data\_postgres\demo\_small.sql – путь к папке «data\_postgres», где хранится бэкап нашей базы «demo\_small.sql»

Данная - локальная (резервная) база данных пригодится нам для работы в любых условиях без подключения к сети и интернет. Базы большого размера позволят почувствовать, как ведут себя запросы на больших объемах данных, и задуматься об оптимизации.

# Отображение локальной базы в Dbeaver 6.1.1

The screenshot displays the DBeaver 6.1.1 interface with a PostgreSQL database connection named "PostgreSQL - demo 2" selected. The left sidebar shows the database structure, including the "demo" schema and its tables. The main workspace shows a diagram of the database schema, including tables like "flights\_v", "routes", "flights", "airports", "aircrafts", "seats", "tickets", "bookings", "boarding\_passes", and "ticket\_flights". The right sidebar shows the "Настройка соединения" (Connection Settings) dialog for "PostgreSQL - demo 2", with fields for Host (localhost), Port (5432), Database (demo), User (postgres), and Password. The "Локальный клиент" (Local client) is set to "PostgreSQL 11". The "Настройки" (Settings) section includes checkboxes for "Показать все базы данных" (Show all databases) and "Показать шаблонные базы данных" (Show template databases). The "Драйвер" (Driver) is set to "PostgreSQL". The "Тест соединения" (Test connection) button is visible. A small "Успешно" (Success) dialog box is also present, indicating the connection was successful.

10 objects

PostgreSQL - demo 2

MSK ru



# Отображение локальной базы в pgAdmin4.

The screenshot displays the pgAdmin 4 web interface in a browser window. The address bar shows the URL `127.0.0.1:54343 pgAdmin 4`. The interface is divided into a left sidebar (Browser) and a main content area (Dashboard).

**Browser (Left Sidebar):**

- Servers (1)
  - PostgreSQL 11
    - Databases (3)
      - TBAPuHA
      - demo (selected)
    - Casts
    - Catalogs (2)
    - Event Triggers
    - Extensions
    - Foreign Data Wrappers
    - Languages
    - Schemas (2)
      - bookings
        - Collations
        - Domains
        - FTS Configurations
        - FTS Dictionaries
        - FTS Parsers
        - FTS Templates
        - Foreign Tables
        - Functions
        - Materialized Views
        - Procedures
        - Sequences
        - Tables (8)
          - aircrafts
          - airports
          - boarding\_passes
          - bookings
          - flights
          - seats
          - ticket\_flights
          - tickets

**Dashboard (Main Content Area):**

The dashboard provides a comprehensive overview of the database's health and performance. It includes several key metrics and charts:

- Database sessions:** A line chart showing the number of total, active, and idle database sessions over time. The y-axis ranges from 1.00 to 3.00.
- Transactions per second:** A line chart showing the rate of transactions, commits, and rollbacks per second. The y-axis ranges from 0.0 to 6.0.
- Tuples in:** A line chart showing the number of tuples inserted, updated, and deleted. The y-axis ranges from 0.00 to 1.00.
- Tuples out:** A line chart showing the number of tuples fetched and returned. The y-axis ranges from 0 to 20,000.
- Block I/O:** A line chart showing the number of block reads and hits. The y-axis ranges from 0 to 400.
- Server activity:** A table showing the current state of the database server, including sessions, locks, and prepared transactions.

**Server activity Table:**

		PID	User	Application	Client	Backend start	State	Wait event	Blocking PIDs
✖	■	▶ 11148	postgres	pgAdmin 4 - DB:demo	::1	2019-08-06 17:45:12 MSK	active		
✖	■	▶ 12076	postgres	DBeaver 6.1.1 - Main	127.0.0.1	2019-08-06 18:05:53 MSK	idle	Client: ClientRead	
✖	■	▶ 12140	postgres	DBeaver 6.1.1 - Metadata	127.0.0.1	2019-08-06 18:05:53 MSK	idle	Client: ClientRead	