МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГТУ», ВГТУ)

Факультет информационных технологий и компьютерной безопасности

# Кафедра автоматизированных и вычислительных систем

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №9

По дисциплине: «Системы хранения и обработки данных»

Тема: «Формирование запросов к базе данных»

Выполнил работу студент группы змИИВТ-241: Матыкина А.В.

подпись, дата

Принял: Короленко В.В.

подпись, дата

Воронеж 2024

# **Учебная задача**

Сформировать SQL-запросы для создания таблиц в СУБД Postgres в соответствии со следующей структурой:

a) Customers — информация о клиентах:

· CustomerID (int, PK) — уникальный идентификатор клиента.

· FirstName (varchar) — имя клиента.

· LastName (varchar) — фамилия клиента.

· Email (varchar) — адрес электронной почты клиента.

b) Orders — информация о заказах:

· OrderID (int, PK) — уникальный идентификатор заказа.

· CustomerID (int) — идентификатор клиента, совершившего заказ.

· OrderDate (datetime) — дата и время создания заказа.

· TotalAmount (decimal) — общая стоимость заказа.

c) OrderDetails — информация о деталях заказов:

· OrderDetailID (int, PK) — уникальный идентификатор детали заказа.

· OrderID (int) — идентификатор заказа.

· ProductID (int) — идентификатор продукта в заказе.

· Quantity (int) — количество продуктов в заказе.

· UnitPrice (decimal) — цена за единицу продукта.

CREATE TABLE IF NOT EXISTS Customers (

CustomerID SERIAL PRIMARY KEY,

FirstName varchar(255),

LastName varchar(255),

Email varchar(255)

);

CREATE TABLE IF NOT EXISTS Orders (

OrderID SERIAL PRIMARY KEY,

CustomerID int REFERENCES Customers (CustomerID),

OrderDate timestamp without time zone,

TotalAmount decimal

);

CREATE TABLE IF NOT EXISTS Products (

ProductID SERIAL PRIMARY KEY,

Product\_name varchar(255)

);

CREATE TABLE IF NOT EXISTS OrderDetails (

OrderDetailID SERIAL PRIMARY KEY,

OrderID int REFERENCES Orders (OrderID),

ProductID int REFERENCES Products (ProductID),

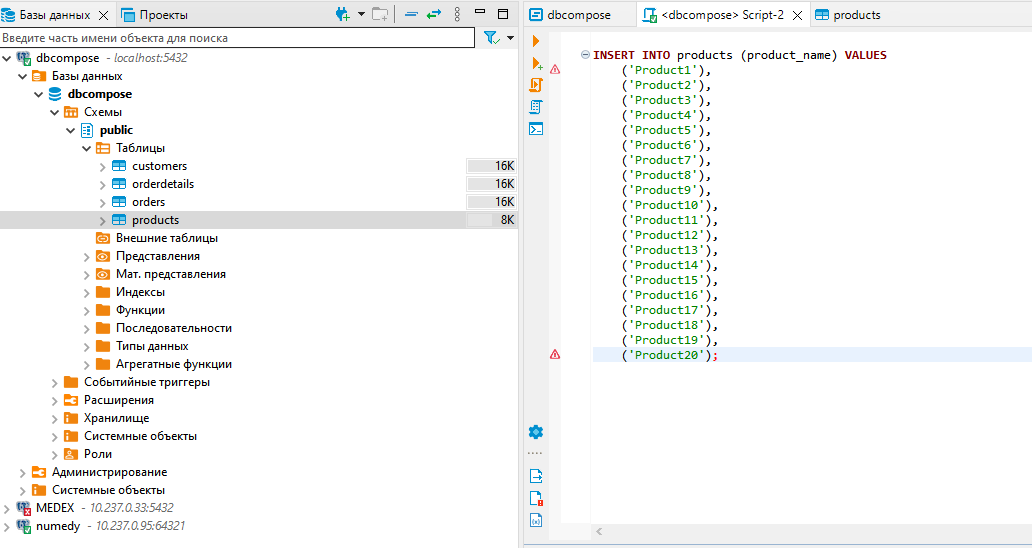
Quantity int,

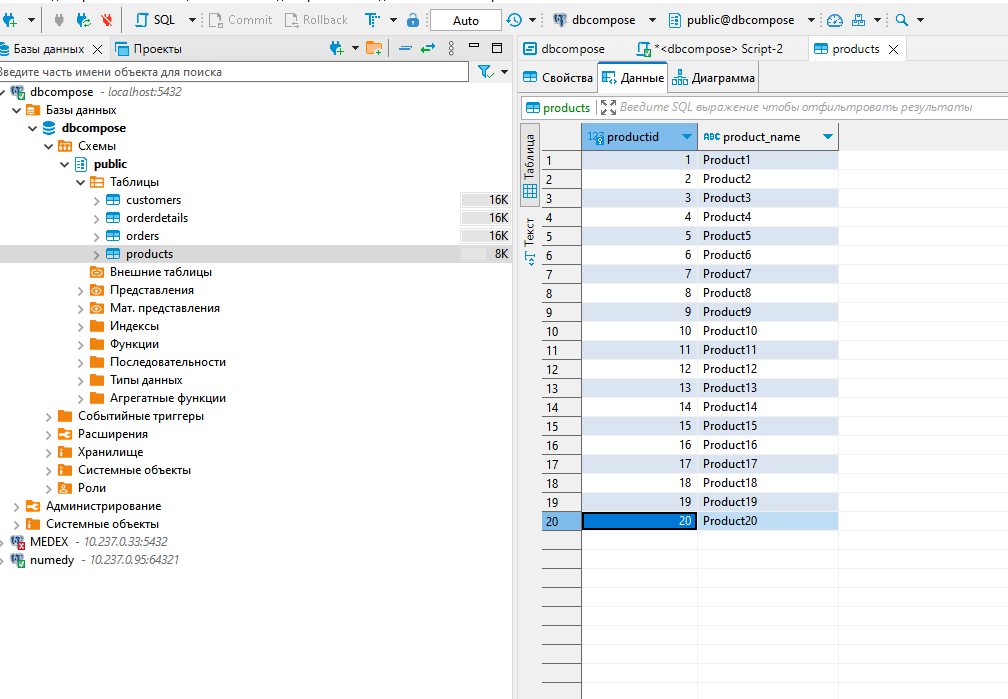
UnitPrice decimal

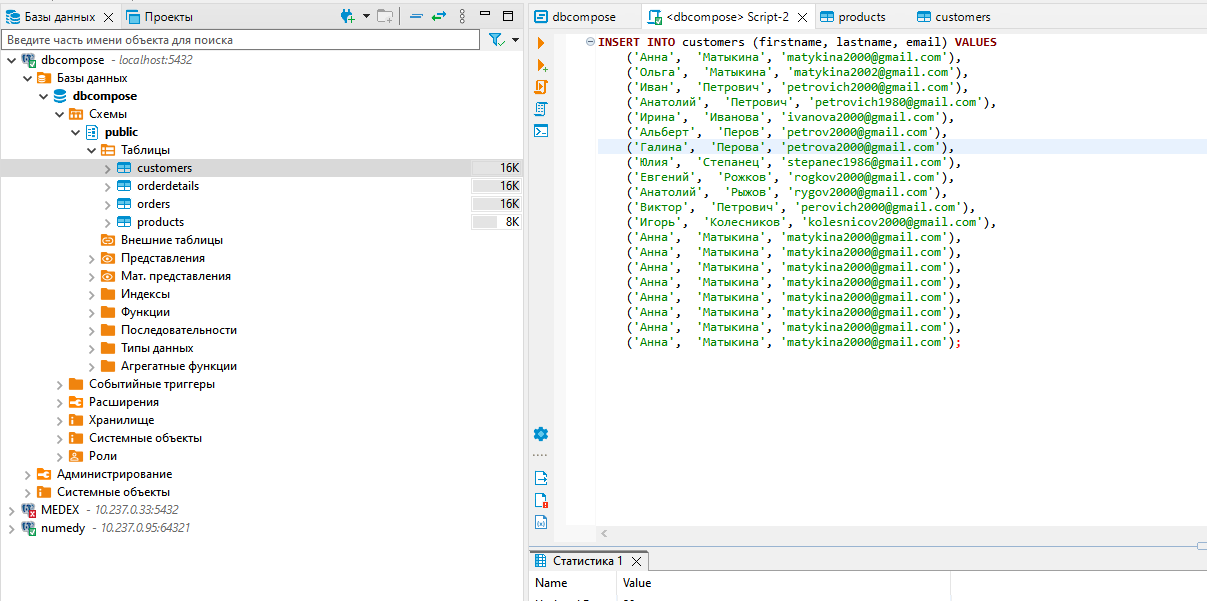
);

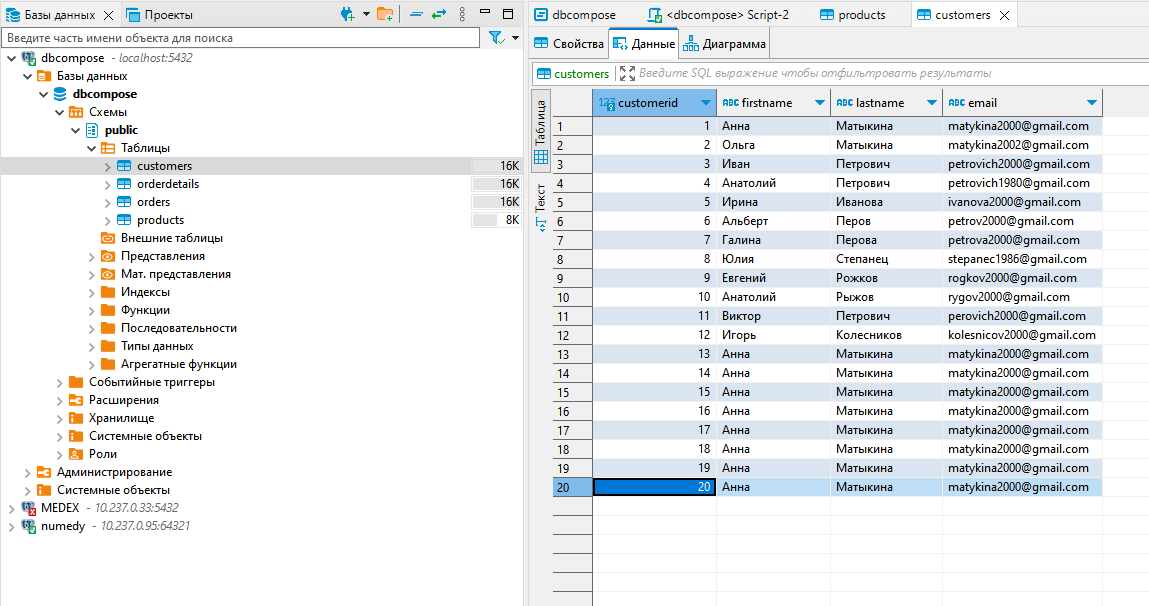


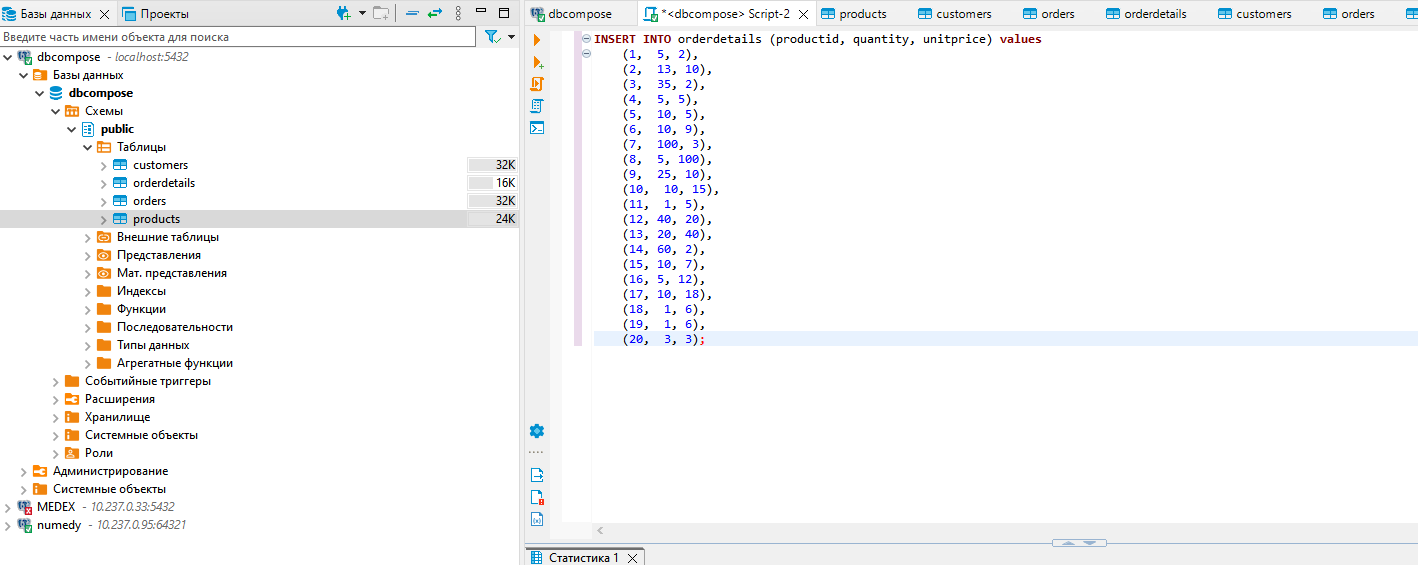
Наполнить базу данных данными (таблицы по 20 – 50 строк)

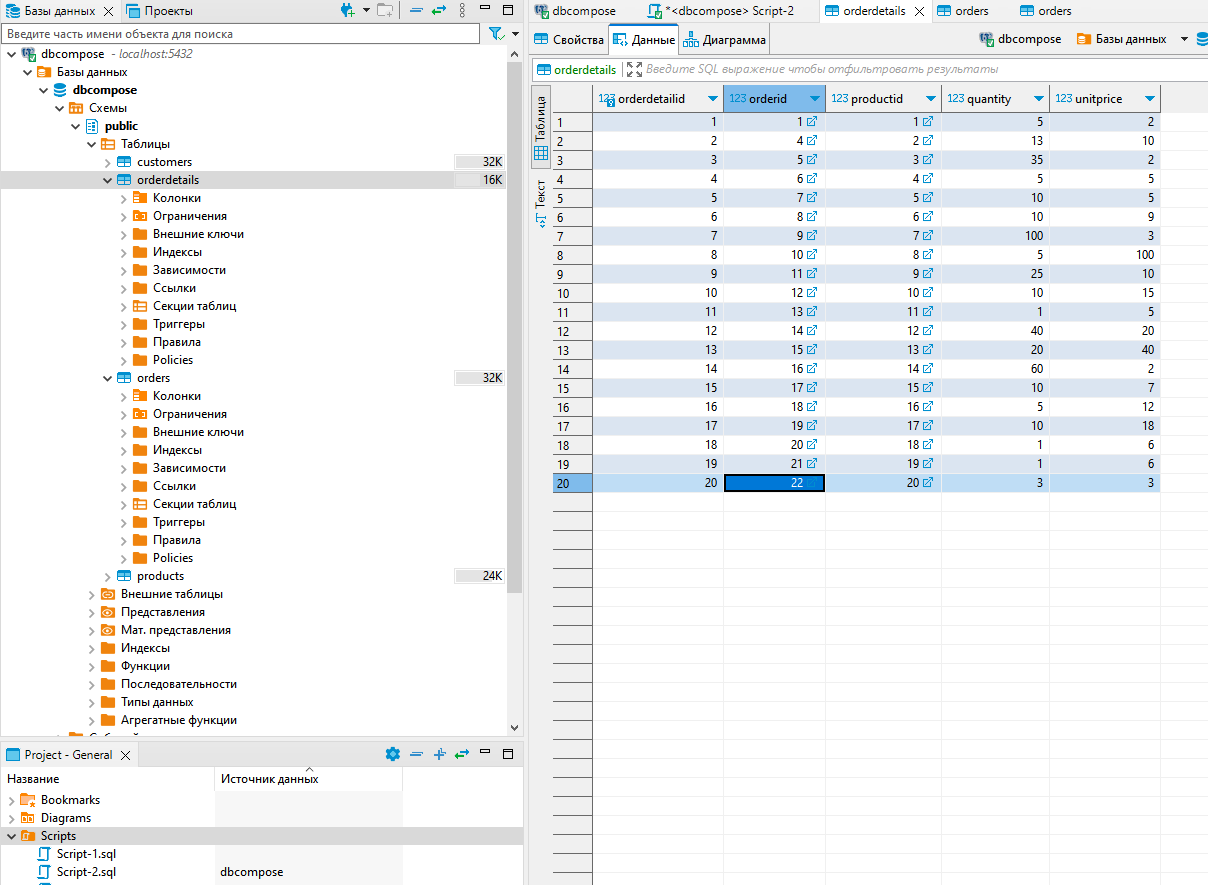












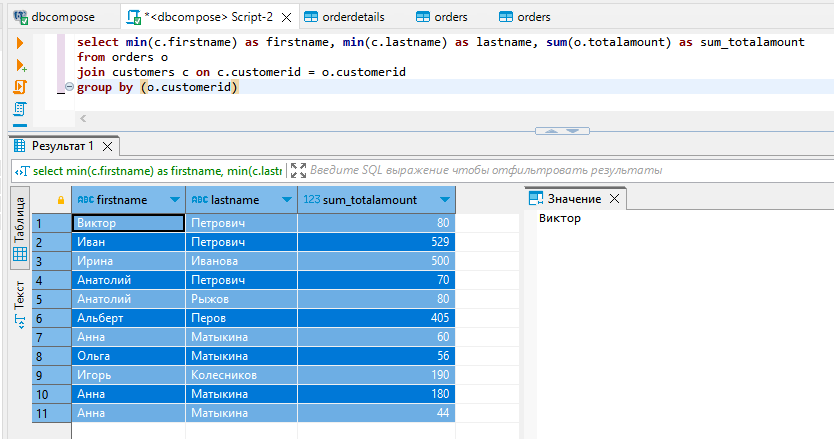
Сформировать SQL-запрос, который возвращает список клиентов (имя и фамилия) и суммарную стоимость заказов каждого клиента (должны отобразиться столбцы с именем, фамилией и стоимостью). То есть все заказы каждого клиента надо просуммировать.

**select** **min**(c.firstname) **as** firstname, **min**(c.lastname) **as** lastname, **sum**(o.totalamount) **as** sum\_totalamount

**from** orders o

**join** customers c **on** c.customerid = o.customerid

**group** **by** (o.customerid)



Отсортировать с полученный список по убыванию суммарной стоимости заказов клиента

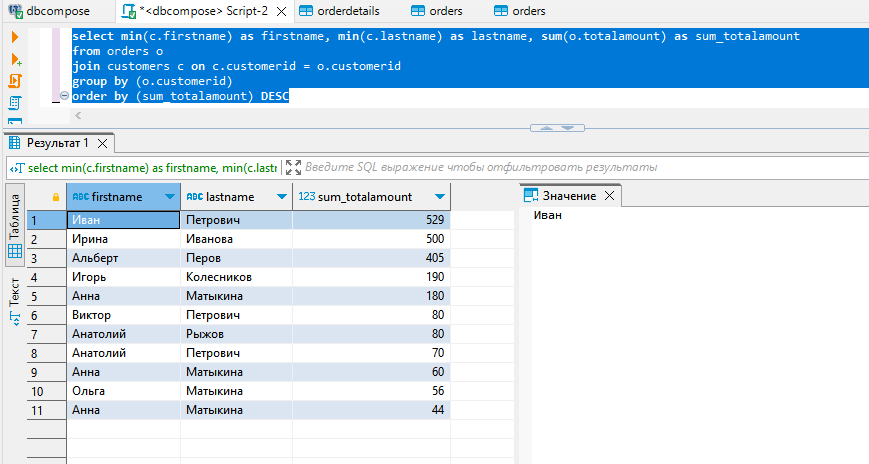
**select** **min**(c.firstname) **as** firstname, **min**(c.lastname) **as** lastname, **sum**(o.totalamount) **as** sum\_totalamount

**from** orders o

**join** customers c **on** c.customerid = o.customerid

**group** **by** (o.customerid)

**order** **by** (sum\_totalamount) **DESC**



Добавить столбец со средней суммарной стоимостью заказов

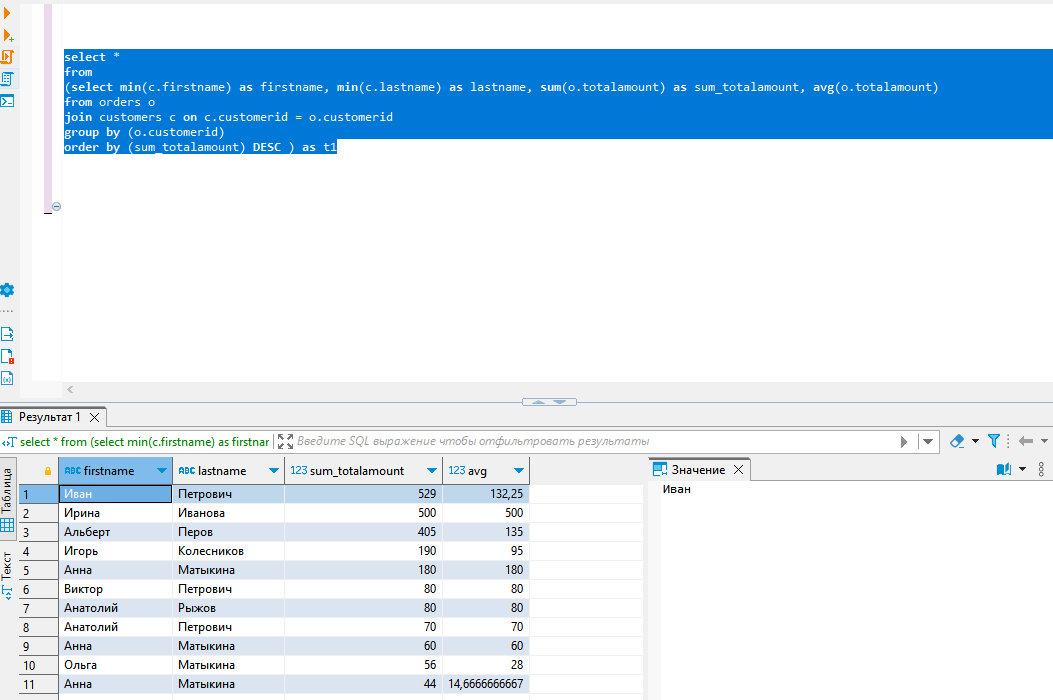
**select** **min**(c.firstname) **as** firstname, **min**(c.lastname) **as** lastname, **sum**(o.totalamount) **as** sum\_totalamount, **avg**(o.totalamount)

**from** orders o

**join** customers c **on** c.customerid = o.customerid

**group** **by** (o.customerid)

**order** **by** (sum\_totalamount) **DESC**



Вывести клиента (список клиентов) с наибольшей суммарной стоимостью заказов (имя, фамилия, стоимость)

**select** t1.firstname, t1.lastname, t1.sum\_totalamount

**from**

(**select** '1' **as** id, **min**(c.firstname) **as** firstname, **min**(c.lastname) **as** lastname, **sum**(o.totalamount) **as** sum\_totalamount

**from** orders o

**join** customers c **on** c.customerid = o.customerid

**group** **by** (o.customerid)

**order** **by** (sum\_totalamount) ) **as** t1

**JOIN**

(**select** '1' **as** id, **max**(t2.sum\_totalamount) **as** max\_sum\_sum\_totalamount

**from**

(**select** **min**(c.firstname) **as** firstname, **min**(c.lastname) **as** lastname, **sum**(o.totalamount) **as** sum\_totalamount

**from** orders o

**join** customers c **on** c.customerid = o.customerid

**group** **by** (o.customerid)

**order** **by** (sum\_totalamount) ) **as** t2) **as** t3

**on** t1.id = t3.id

**where** t3.max\_sum\_sum\_totalamount = t1.sum\_totalamount

Для каждого клиента с наибольшей суммарной стоимостью заказов вывести список его заказов (номер заказа и стоимость) в порядке возрастания стоимости заказа

**select** o.orderid, o.totalamount

**from**

(**select** t1.customerid, t1.firstname, t1.lastname, t1.sum\_totalamount

**from**

(**select** '1' **as** id, o.customerid, **min**(c.firstname) **as** firstname, **min**(c.lastname) **as** lastname, **sum**(o.totalamount) **as** sum\_totalamount

**from** orders o

**join** customers c **on** c.customerid = o.customerid

**group** **by** (o.customerid)

**order** **by** (sum\_totalamount) ) **as** t1

**join**

(**select** '1' **as** id, **max**(t2.sum\_totalamount) **as** max\_sum\_sum\_totalamount

**from**

(**select** **min**(c.firstname) **as** firstname, **min**(c.lastname) **as** lastname, **sum**(o.totalamount) **as** sum\_totalamount

**from** orders o

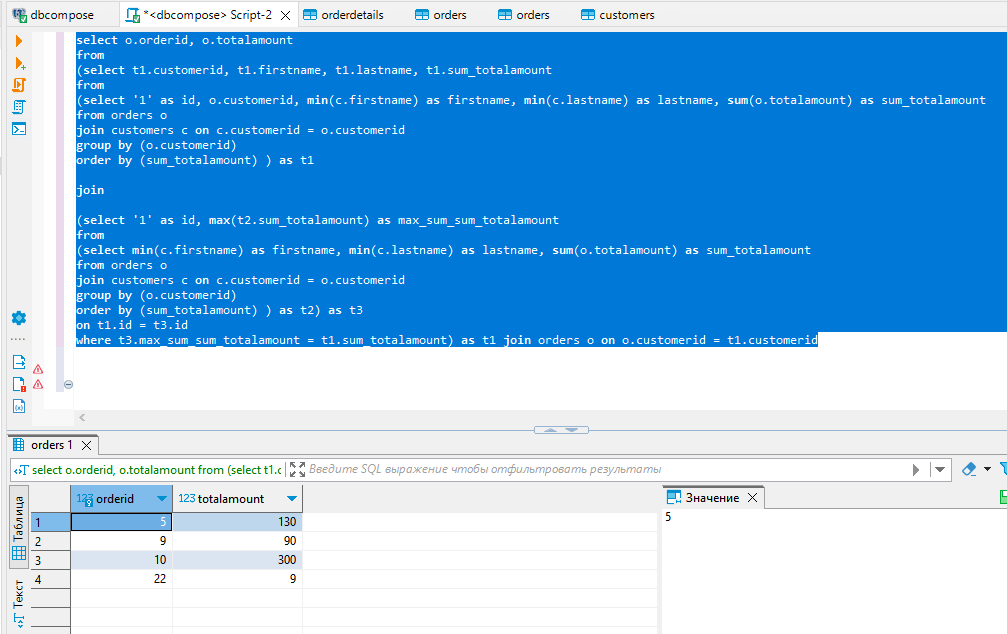
**join** customers c **on** c.customerid = o.customerid

**group** **by** (o.customerid)

**order** **by** (sum\_totalamount) ) **as** t2) **as** t3

**on** t1.id = t3.id

**where** t3.max\_sum\_sum\_totalamount = t1.sum\_totalamount) **as** t1 **join** orders o **on** o.customerid = t1.customerid



Вывести только тех клиентов, у которых суммарная стоимость заказов превышает среднюю суммарную стоимость заказов клиентов (имя, фамилия, суммарная стоимость заказов клиента, средняя стоимость заказа)

**select** t2.firstname, t2.lastname, t2.sum\_totalamount, t1.avg\_sum\_totalamount

**from**

(**select** '1' **as** id, **min**(c.firstname) **as** firstname, **min**(c.lastname) **as** lastname, **sum**(o.totalamount) **as** sum\_totalamount

**from** orders o

**join** customers c **on** c.customerid = o.customerid

**group** **by** (o.customerid)

**order** **by** (sum\_totalamount)) **as** t2

**join**

(**select** '1'**as** id, **avg**(t1.sum\_totalamount) **as** avg\_sum\_totalamount **from**

(**select** **sum**(o.totalamount) **as** sum\_totalamount

**from** orders o

**join** customers c **on** c.customerid = o.customerid

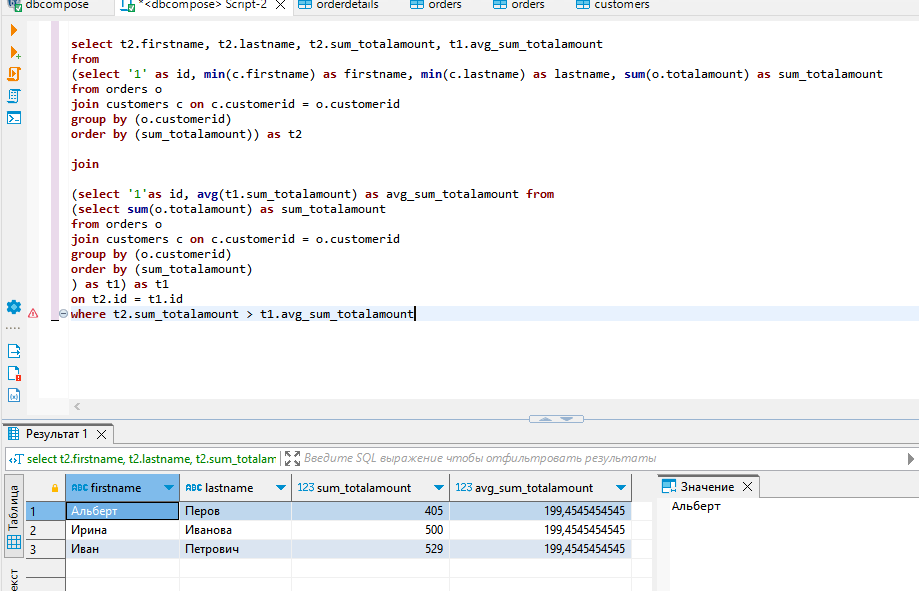
**group** **by** (o.customerid)

**order** **by** (sum\_totalamount)

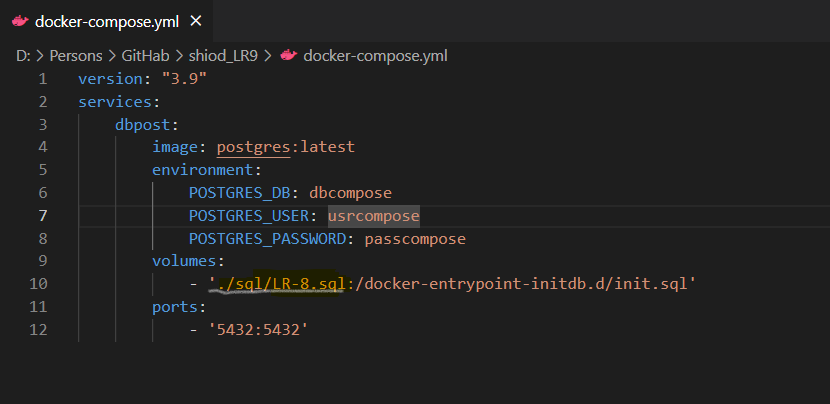
) **as** t1) **as** t1

**on** t2.id = t1.id

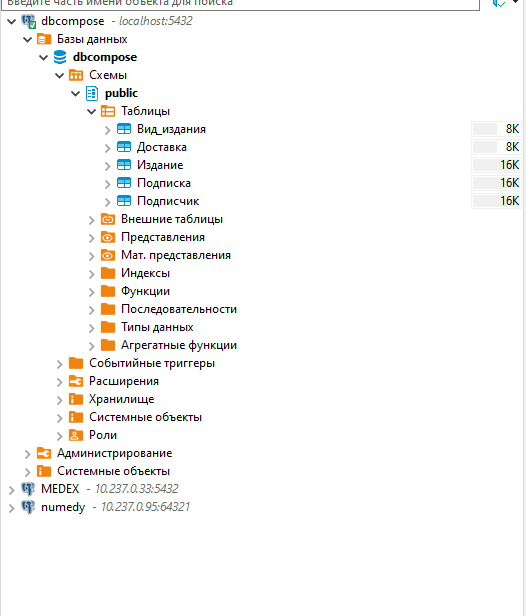
**where** t2.sum\_totalamount > t1.avg\_sum\_totalamount



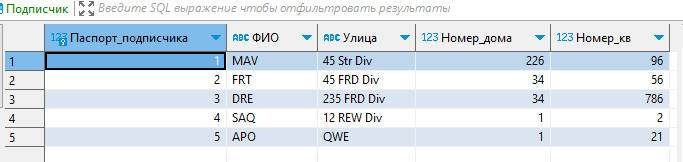
Для создания нового соединения с новой БД переписываю docker-compose.yml

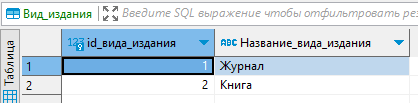


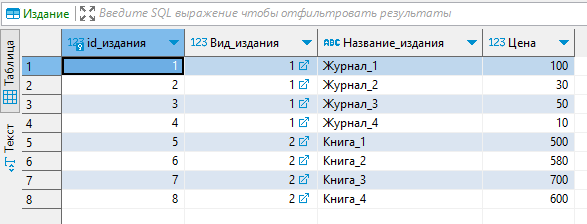
В результате видим новую БД:

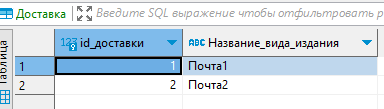


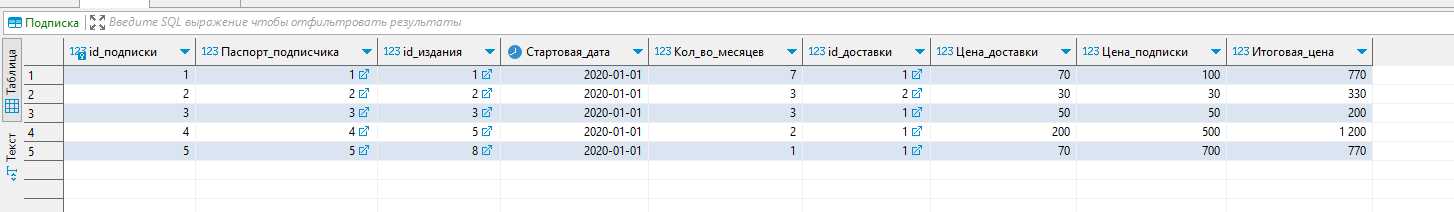
Заполняем таблицы данными:











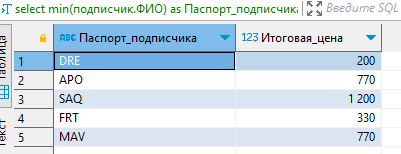
Сформировать SQL-запрос, который возвращает список подписчиков (ФИО) и суммарную стоимость заказов каждого клиента (должны отобразиться столбцы с ФИО и стоимостью). То есть все заказы каждого подписчика надо просуммировать.

**select** **min**(подписчик.ФИО) **as** Паспорт\_подписчика, **sum**(п.Итоговая\_цена) **as** Итоговая\_цена

**from** Подписка п

**join** Подписчик подписчик **on** п.Паспорт\_подписчика = подписчик.Паспорт\_подписчика

**group** **by** (п.Паспорт\_подписчика)



Отсортировать с полученный список по убыванию суммарной стоимости заказов подписчика

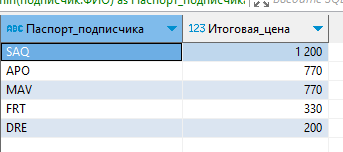
**select** **min**(подписчик.ФИО) **as** Паспорт\_подписчика, **sum**(п.Итоговая\_цена) **as** Итоговая\_цена

**from** Подписка п

**join** Подписчик подписчик **on** п.Паспорт\_подписчика = подписчик.Паспорт\_подписчика

**group** **by** (п.Паспорт\_подписчика)

**order** **by** (Итоговая\_цена) **DESC**

****

Добавить столбец со средней суммарной стоимостью заказов

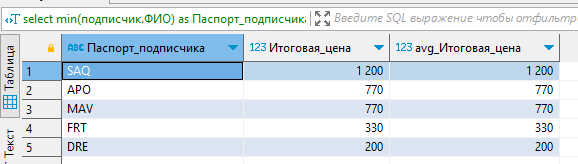
**select** **min**(подписчик.ФИО) **as** Паспорт\_подписчика, **sum**(п.Итоговая\_цена) **as** Итоговая\_цена, **avg**(п.Итоговая\_цена) **as** avg\_Итоговая\_цена

**from** Подписка п

**join** Подписчик подписчик **on** п.Паспорт\_подписчика = подписчик.Паспорт\_подписчика

**group** **by** (п.Паспорт\_подписчика)

**order** **by** (Итоговая\_цена) **DESC**



Вывести подписчика с наибольшей суммарной стоимостью заказов (ФИО, стоимость)

**select** t1.ФИО

**from**

(**select** '1' **as** id, **min**(подписчик.ФИО) **as** ФИО, **sum**(п.Итоговая\_цена) **as** Итоговая\_цена

**from** Подписка п

**join** Подписчик подписчик **on** п.Паспорт\_подписчика = подписчик.Паспорт\_подписчика

**group** **by** (п.Паспорт\_подписчика) )**as** t1

**join**

(

**select** '1' **as** id, **max**(t1.Итоговая\_цена) **as** Итоговая\_цена

**from**

(

**select** **sum**(п.Итоговая\_цена) **as** Итоговая\_цена

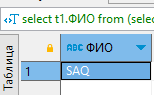
**from** Подписка п

**join** Подписчик подписчик **on** п.Паспорт\_подписчика = подписчик.Паспорт\_подписчика

**group** **by** (п.Паспорт\_подписчика) ) **as** t1) **as** t2

**on** t1.id = t2.id

**where** t1.Итоговая\_цена = t2.Итоговая\_цена



Для каждого подписчика с наибольшей суммарной стоимостью заказов вывести список его заказов (номер заказа и стоимость) в порядке возрастания стоимости заказа

**select** п.id\_подписки, п.Итоговая\_цена

**from**

(**select** t1.ФИО, t1.Паспорт\_подписчика

**from**

(**select** '1' **as** id, п.Паспорт\_подписчика **as** Паспорт\_подписчика, **min**(подписчик.ФИО) **as** ФИО, **sum**(п.Итоговая\_цена) **as** Итоговая\_цена

**from** Подписка п

**join** Подписчик подписчик **on** п.Паспорт\_подписчика = подписчик.Паспорт\_подписчика

**group** **by** (п.Паспорт\_подписчика) )**as** t1

**join**

(

**select** '1' **as** id, **max**(t1.Итоговая\_цена) **as** Итоговая\_цена

**from**

(

**select** **sum**(п.Итоговая\_цена) **as** Итоговая\_цена

**from** Подписка п

**join** Подписчик подписчик **on** п.Паспорт\_подписчика = подписчик.Паспорт\_подписчика

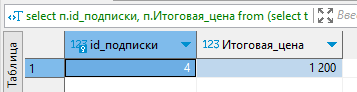
**group** **by** (п.Паспорт\_подписчика) ) **as** t1) **as** t2

**on** t1.id = t2.id

**where** t1.Итоговая\_цена = t2.Итоговая\_цена) **as** t1

**join** Подписка п **on** п.Паспорт\_подписчика = t1.Паспорт\_подписчика

**order** **by** (п.Итоговая\_цена)



# **Контрольные вопросы**

1. Как заполнить таблицу данными?

Использовать команду

INSERT INTO <имя*таблицы> VALUES ('<знач*\_стлб\_1>',*<знач*\_стлб\_2>, '*<знач*\_стлб\_3>', '*<знач*\_стлб\_4>', …), …;

1. Агрегатные функции. Основные сведения.

Агрегатная функция – это функция, которая выполняет вычисление на наборе значений и возвращает одиночное значение.

Чтобы вызвать агрегатную функцию, нужно написать ее имя и затем в круглых скобках передать аргументы — переменные, над которыми будут совершаться действия.

SUM(поле\_таблицы) Возвращает сумму значений

AVG(поле\_таблицы) Возвращает среднее значение

COUNT(поле\_таблицы) Возвращает количество записей

MIN(поле\_таблицы) Возвращает минимальное значение

MAX(поле\_таблицы) Возвращает максимальное значение

1. Как объединить таблицы в запросе?

JOIN — оператор языка SQL, который является реализацией операции соединения реляционной алгебры.

После ключевого слова ON записывается условие объединения таблиц

1. Способы объединения таблиц. Краткая характеристика.

INNER JOIN используется для возвращения только тех строк, которые имеют совпадения в обеих таблицах. Возвращаются только те строки, где ключевые значения совпадают в обеих таблицах.

SELF JOIN — это операция объединения таблицы с самой собой. Это означает, что при SELF JOIN таблица используется дважды в одном запросе. Чтобы не путаться, каждое вхождение таблицы в команде использует псевдоним.

LEFT OUTER JOIN используется для объединения двух таблиц, но с сохранением всех записей из левой таблицы, в том числе и тех, для которых нет соответствующих записей в правой таблице.

RIGHT OUTER JOIN используется для объединения двух таблиц, но для сохранения всех записей из правой таблицы, в том числе и тех, для которых нет соответствующих записей в левой таблице.

FULL OUTER JOIN используется для объединения двух таблиц с сохранением всех записей, включая те, для которых нет соответствующих записей в другой таблице.

1. План запроса. Краткое описание.

Это набор конкретных действий, выполнение которых приведёт SQL-запрос к итоговому результату. Он определяет, как будет осуществляться доступ к исходным данным, в каком порядке, какие конкретные методы будут использоваться для извлечения данных из каждой таблицы, для вычислений, фильтрации, статистической обработки и сортировки данных.

План в целом разделяется на две стадии:

1. Выбор результатов. Выполняется способами: вложенные циклы или слияние.
2. Сортировка и группировка, выполнение агрегаций. Это опциональная стадия, которая выполняется, если не найдено путей доступа для получения результата в запрошенном порядке.
3. Оптимизация запроса. Краткое описание.

Индекс — это дополнительная структура данных, созданная на основе столбцов таблицы. При добавлении индекса СУБД организует данные таким образом, чтобы ускорить операции чтения.

Есть несколько вариантов индексов.

1. Первичные индексы

Первичный индекс создаётся автоматически для каждой таблицы, в которой объявлен PRIMARY KEY. Этот индекс гарантирует уникальность значений и позволяет быстро находить записи по основному идентификатору.

2. Уникальные индексы

Уникальные индексы предотвращают дублирование данных. Например, это актуально для столбцов, в которых должны храниться уникальные значения

3. Составные индексы

Составной индекс создаётся для нескольких столбцов одновременно. Он полезен для ускорения запросов, которые используют фильтры по 2-м и более полям.

1. Оконные функции. Краткое описание.

Оконные функции в SQL — это инструмент, который позволяет выполнять вычисления по набору строк, связанных с текущей строкой. Они часто используются для создания аналитических отчётов и выполнения сложных вычислений, таких как ранжирование, суммирование и т. д.

Суть оконных функций в SQL заключается в разделении запроса на части, или так называемые партиции, по определённым условиям. При этом каждая часть обрабатывается отдельно от других. Функция выдаёт отдельный результат для каждой строки, сохраняя его.

SELECT <оконная\_функция>(<поле\_таблицы>)

OVER (

[PARTITION BY <столбцы\_для\_разделения>]

[ORDER BY <столбцы\_для\_сортировки>]

[ROWS|RANGE <определение\_диапазона\_строк>]

)

1. Команда и синтаксис для группировки данных.

Команда для группировки данных в SQL — GROUP BY.Она позволяет группировать строки в таблице по одному или нескольким столбцам.

Синтаксис: SELECT столбец1, агрегатная\_функция(столбец2) FROM таблица GROUP BY столбец1

1. Сортировка данных. Краткое описание.

Для сортировки данных в SQL используется оператор ORDER BY.

Направления сортировки задаются ключевыми словами:

ASC (от английского ascending — по возрастанию).

DESC (от английского descending — по убыванию).

Синтаксис запроса выглядит следующим образом:

SELECT columns FROM table ORDER BY column [ASC | DESC].

1. Условия для выборки данных (WHERE, HAVING). Краткое описание.

WHERE используется для фильтрации записей до группировки, отсеивая те, которые не удовлетворяют заданному критерию. В свою очередь, HAVING применяется после операции GROUP BY и позволяет отфильтровать группы записей по агрегированным значениям, получаемым с помощью функций SUM, AVG или COUNT.