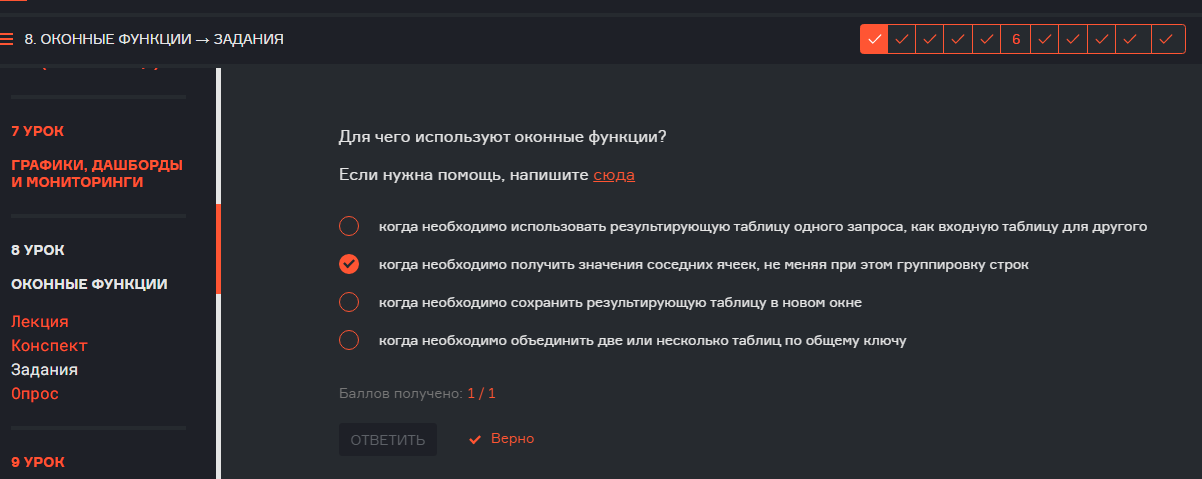
Оконные функции используют, **когда необходимо получить значения соседних ячеек, не меняя при этом группировку строк.**



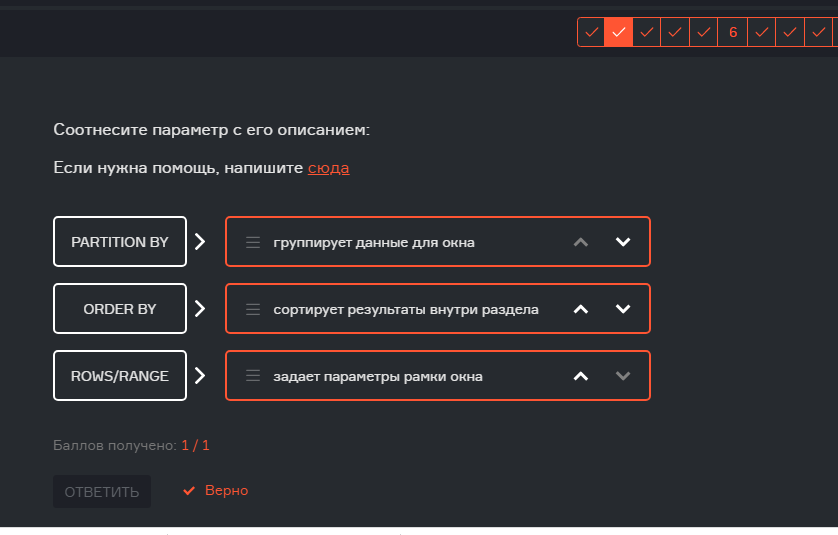
Разбор:

Правильный ответ:

* PARTITION BY – группирует данные для окна
* ORDER BY – сортирует результаты внутри раздела
* ROWS/RANGE – задает параметры рамки окна

Оконные функции могут быть сложны для понимания, особенно часть, касающаяся синтаксиса ROWS/RANGE, но после небольшой практики их использование не составит труда. Попробуйте самостоятельно написать несколько простых запросов с оконными функциями к базе, чтобы привыкнуть к синтаксису.

2.



Разбор:

**ROWS / RANGE**

Задают **параметры** **рамки** окна. Рамки используются в тех оконных функциях, которые работают с рамками, а не с разделом целиком. Первым аргументом задается начало рамки, вторым аргументом конец рамки, и дополнительно, третьим аргументом задается исключение из рамки.

Рамку можно задать в нескольких режимах, а именно:

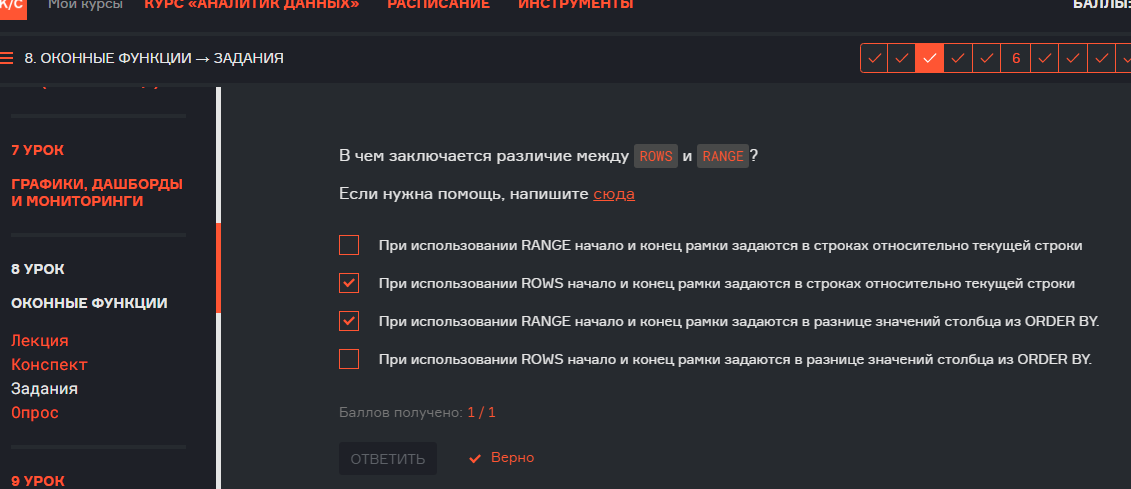
* ROWS – Начало и конец рамки задаются в строках относительно текущей строки. Например ROWS BETWEEN 3 PRECEDING AND 3 FOLLOWING означает создание рамки на 3 строки вверх и вниз относительно текущей строки.
* RANGE – Начало и конец рамки задаются в разнице значений столбца из ORDER BY. Например, если в ORDER BY находится столбец event\_date с типом данных date, то определение окна можно задать следующим образом RANGE BETWEEN '3 day' PRECEDING AND '3 days' FOLLOWING что будет означать рамку на 3 дня назад и вперед.

При указании рамки через RANGE обязательным условием является только один столбец в ORDER BY окна.

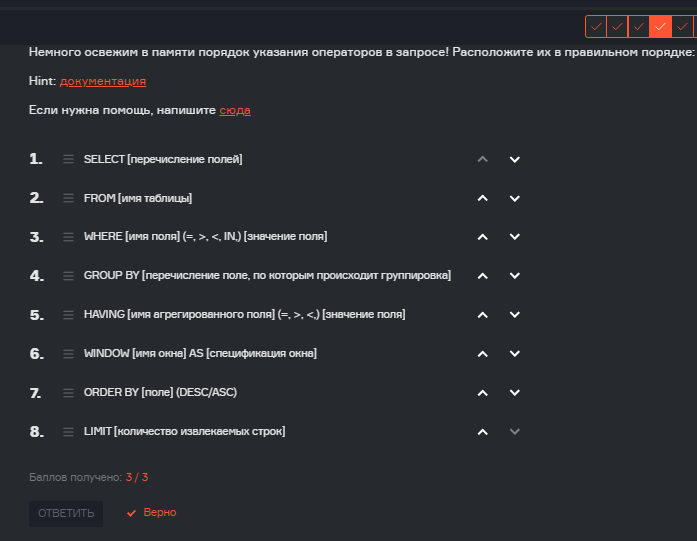
Правильные ответы:

* При использовании ROWS начало и конец рамки задаются в строках относительно текущей строки
* При использовании RANGE начало и конец рамки задаются в разнице значений столбца из ORDER BY.

3



4.



Разбор:

С добавлением оконных функций несколько усложнилась, но к этому необходимо привыкнуть:

SELECT [перечисление полей]

FROM [имя таблицы]

WHERE [имя поля] (=, >, <, IN,) [значение поля]

GROUP BY [перечисление поле, по которым происходит группировка]

HAVING [имя агрегированного поля] (=, >, <,) [значение поля]

WINDOW [имя окна] AS [спецификация окна]

ORDER BY [поле] (DESC/ASC)

LIMIT [количество извлекаемых строк]

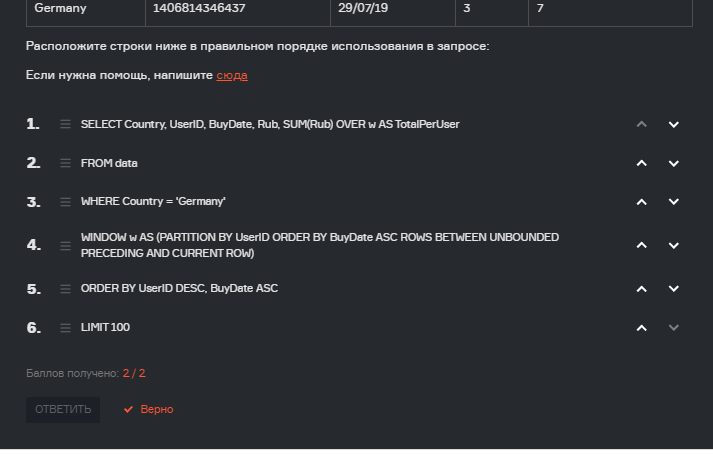
5

Посмотрим на пример. Предположим, есть таблица с пользователями, состоящая из 4 колонок. Для пользователей из Германии нужно посчитать накопительную сумму покупок и записать в TotalPerUser. Например:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Country** | **UserID** | **BuyDate** | **Rub** | **TotalPerUser** |
| Germany | 1406814346437 | 27/07/19 | 4 | 4 |
| Germany | 1406814346437 | 29/07/19 | 3 | 7 |

Расположите строки ниже в правильном порядке использования в запросе:

Если нужна помощь, напишите [сюда](https://1510karpovanalytics.slack.com/archives/C02HN0VB8HY/p1634240309048500?thread_ts=1634240309.048400&cid=C02HN0VB8HY)



Разбор:

Более конкретно познакомиться со структурой можно на примере запроса из базы данных.

Правильный ответ:

SELECT Country, UserID, BuyDate, Rub, SUM(Rub) OVER w AS TotalPerUser

FROM data

WHERE Country = 'Germany'

WINDOW w AS

(PARTITION BY UserID

ORDER BY BuyDate ASC

ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)

ORDER BY

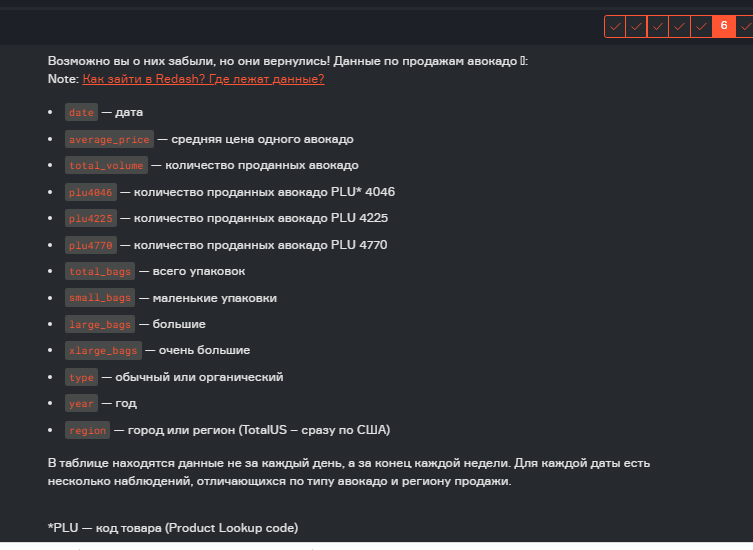
UserID DESC,

BuyDate ASC

LIMIT 100

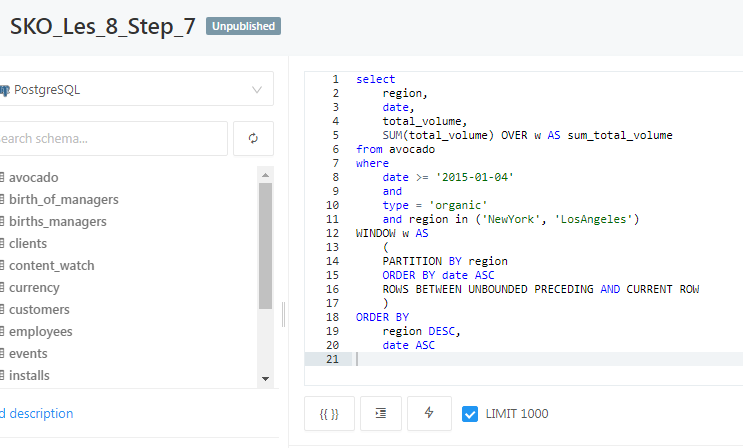
6.

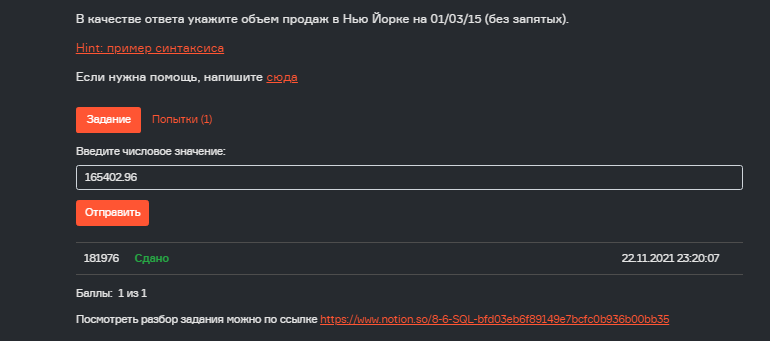
Описание:



7.







Разбор:

Какой алгоритм можно использовать для решения подобных задач?

1. в задаче требуется посчитать кумулятивную сумму проданных товаров на каждую дату - то есть, для каждой последующей строки ее значение нужно складывать с предыдущей. Так как нам нужно видеть этот показатель для каждой строки исходной таблицы, используем оконную функцию, которая не поменяет ничего в исходной таблице;
2. из измерений нас интересует только регион - значит, в нашей оконной функции будет использоваться PARTITION BY region, а ORDER BY - по дате, причем в порядке возрастания
3. чтобы для каждой строки посчитать ее сумму и сумму всех предыдущих (=более ранних по дате, как задано ORDER BY), без каких-либо более сложных сдвигов по окну, - используем ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

Таким образом, наша оконная функция будет выглядеть так:

WINDOW w AS (

PARTITION BY region

ORDER BY date ASC

ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

)

Так мы описываем, с какими строками будет работать агрегатная функция в блоке SELECT. Саму функцию зададим следующим образом:

SUM(total\_volume) OVER w AS volume

Пример корректного запроса:

SELECT

region,

date,

total\_volume,

SUM(total\_volume) OVER w AS volume

FROM

avocado

WHERE

region IN ('NewYork', 'LosAngeles') AND type='organic'

WINDOW w AS (

PARTITION BY region

ORDER BY date ASC

ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

)

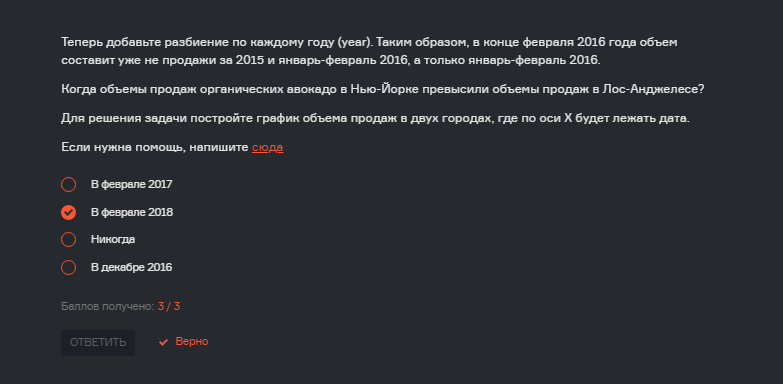
ORDER BY

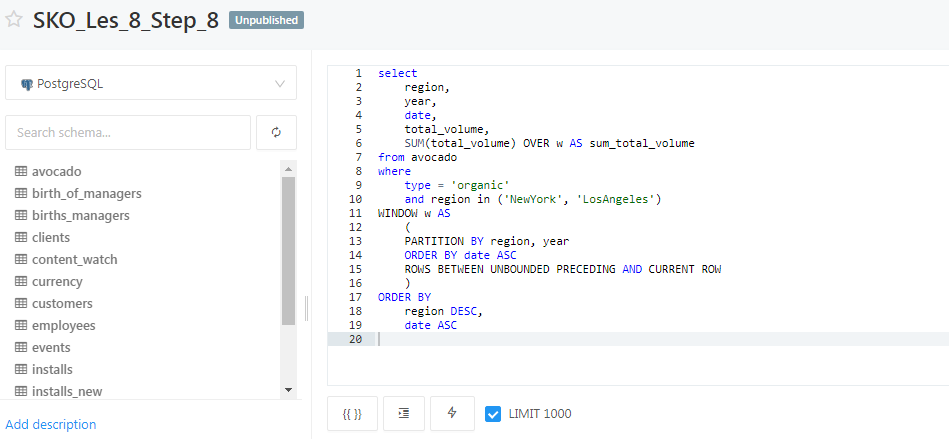
region DESC,

date

**Правильный ответ:**165402.96

8.





Разбор:

В данном задании нужно модифицировать предыдущий запрос, добавив в оконную функцию PARTITION BY region, year - этого будет достаточно, чтобы получить разбиение по году. Также рекомендуется визуализировать результат в Redash - так ответ будет очевиден

Пример корректного запроса:

SELECT

region,

date,

total\_volume,

SUM(total\_volume) OVER w AS volume

FROM

avocado

WHERE

region IN ('NewYork', 'LosAngeles') AND type='organic'

WINDOW w AS (

PARTITION BY region, year

ORDER BY date ASC

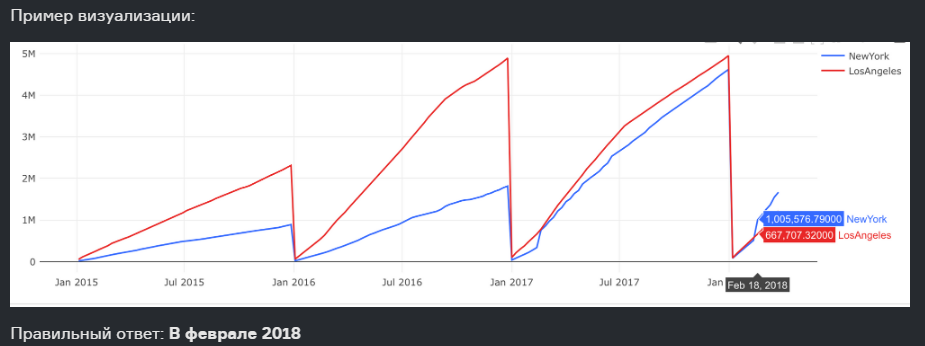
ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

)

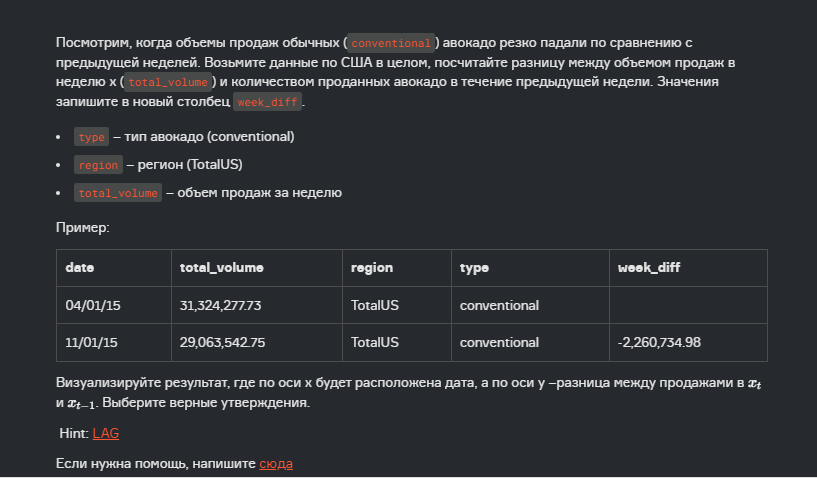
ORDER BY

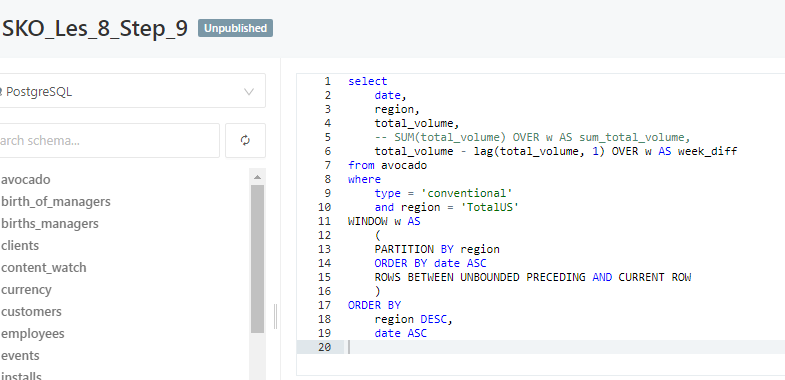
region DESC,

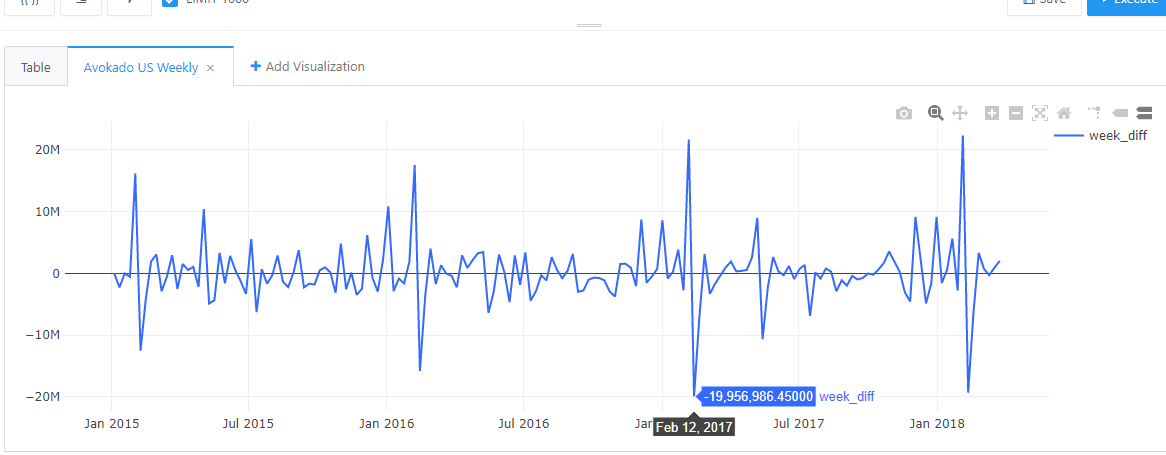
date

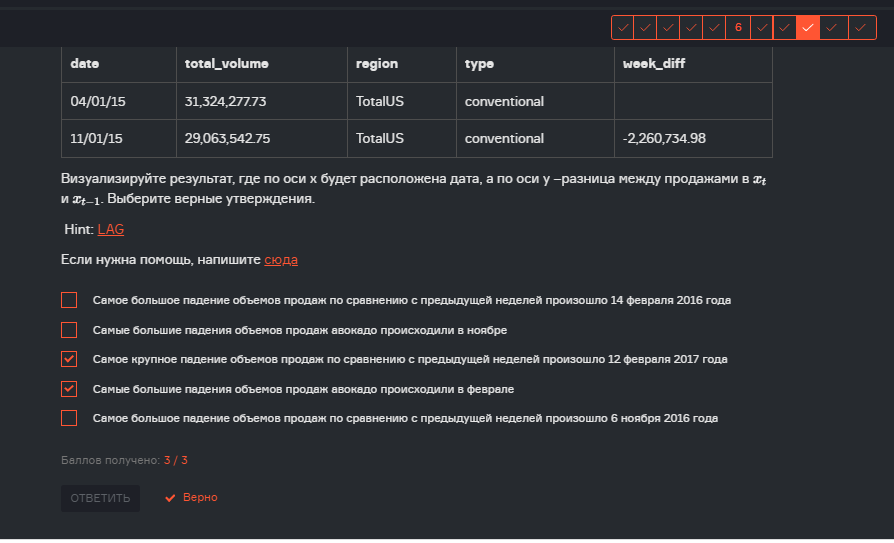


9.









Разбор:

Это задание, как и предыдущее, лучше всего сделать сразу в Redash - на графике ответ на задачу будет очевиден.

Сначала спроектируем запрос:

Посмотрим, какие именно данные нам нужны — это данные о продажах обычных авокадо в США, то есть, фильтры будут такие:

WHERE

region = 'TotalUS'

AND type = 'conventional'

В окне нам не будут необходимы дополнительные партиции по этому достаточно оставить только сортировку и параметры

WINDOW w AS (

    ORDER BY date ASC

    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

)

 В SELECT передадим разность между **total\_volume**и LAG(total\_volume, 1)поверх нашего окна w, с её помощью мы сможем вычитать предыдущую строчку

Пример корректного запроса:

SELECT

    date,

    total\_volume,

    region,

    total\_volume - LAG(total\_volume, 1) over w as week\_diff

FROM

    avocado

WHERE

    region = 'TotalUS' AND type='conventional'

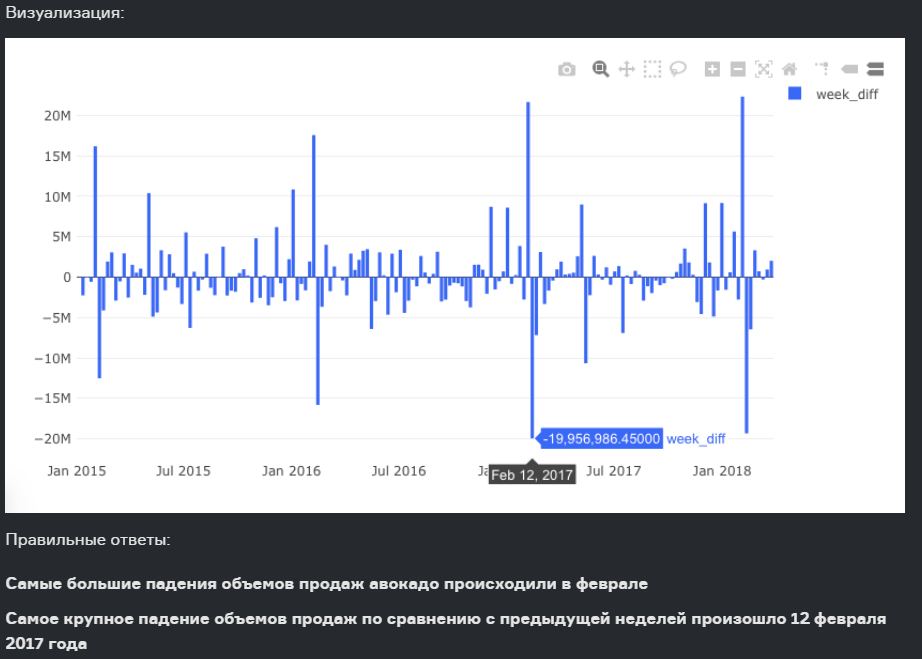
WINDOW w AS (

    ORDER BY date ASC

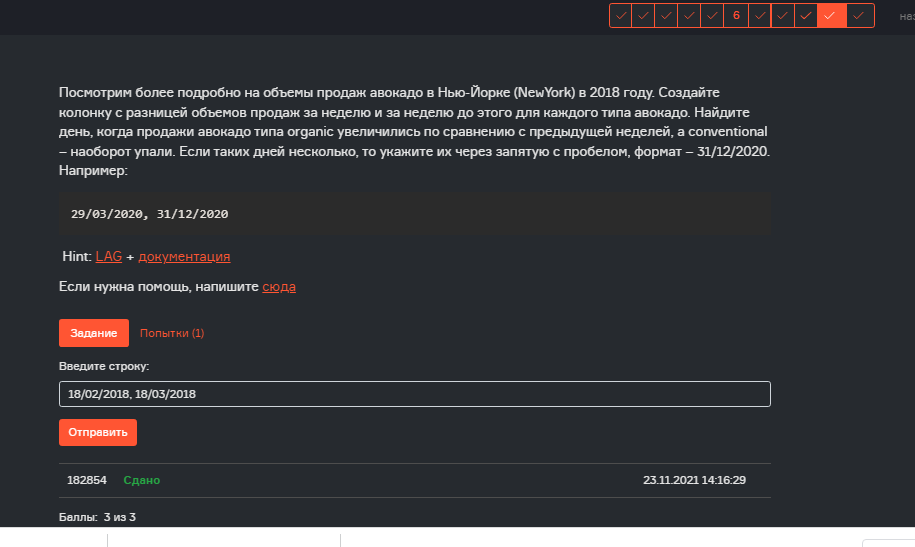
    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

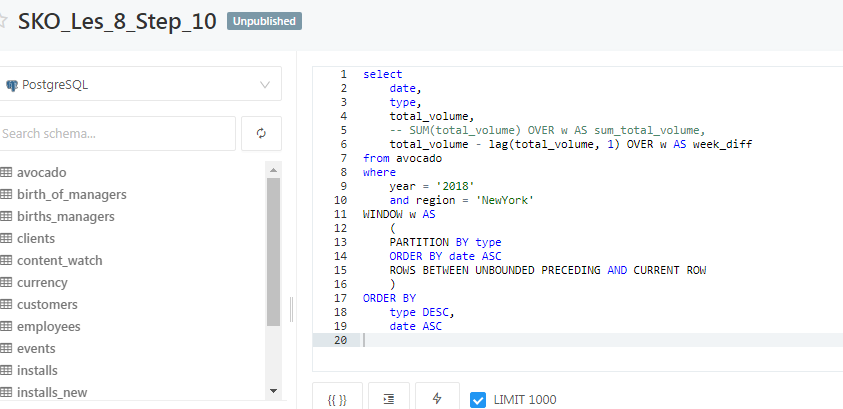
)

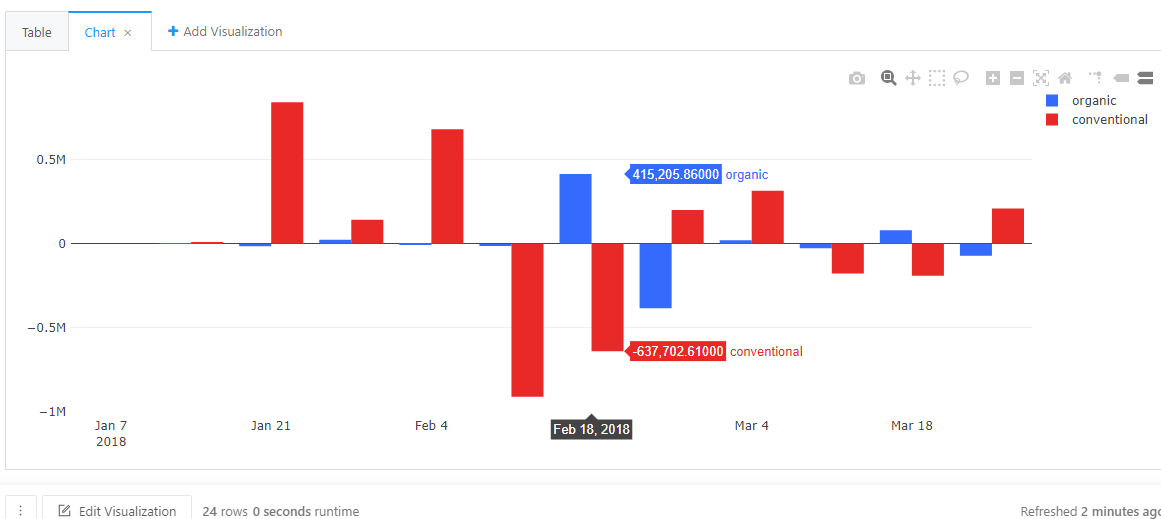
ORDER BY  date



10







Разбор:

Подготовим запрос.

Теперь нам интересен только New-York за 2018 год, так как теперь нам важно смотреть на разницу в продаже между типами авокадо, мы добавляем в партицию для окна колонку **type**:

WINDOW w AS (

PARTITION BY type

ORDER BY date ASC

ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

)

Пример корректного запроса:

SELECT

type,

date,

total\_volume - LAG(total\_volume, 1) over w as week\_diff

FROM

avocado

WHERE

region = 'NewYork' and year = 2018

WINDOW w AS (

PARTITION BY type

ORDER BY date ASC

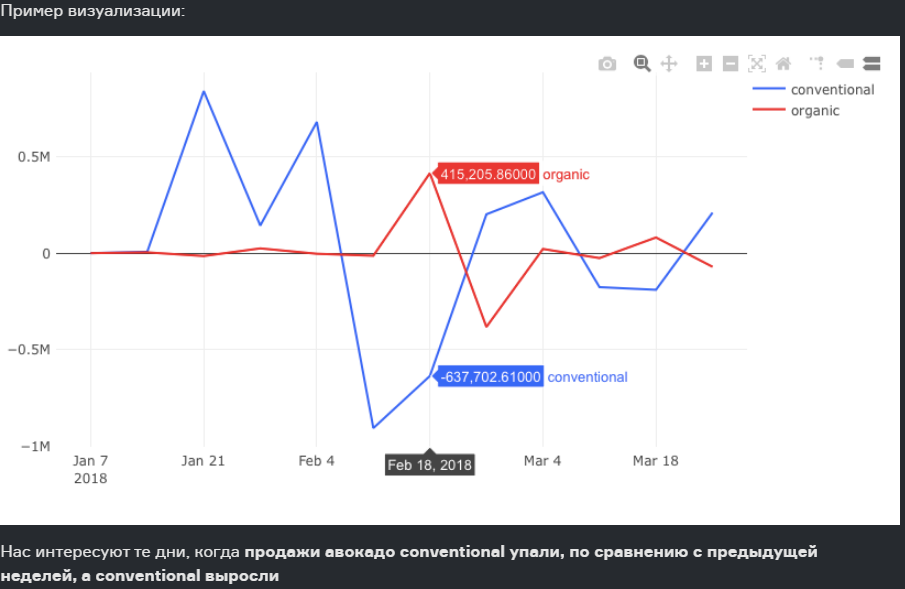
ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

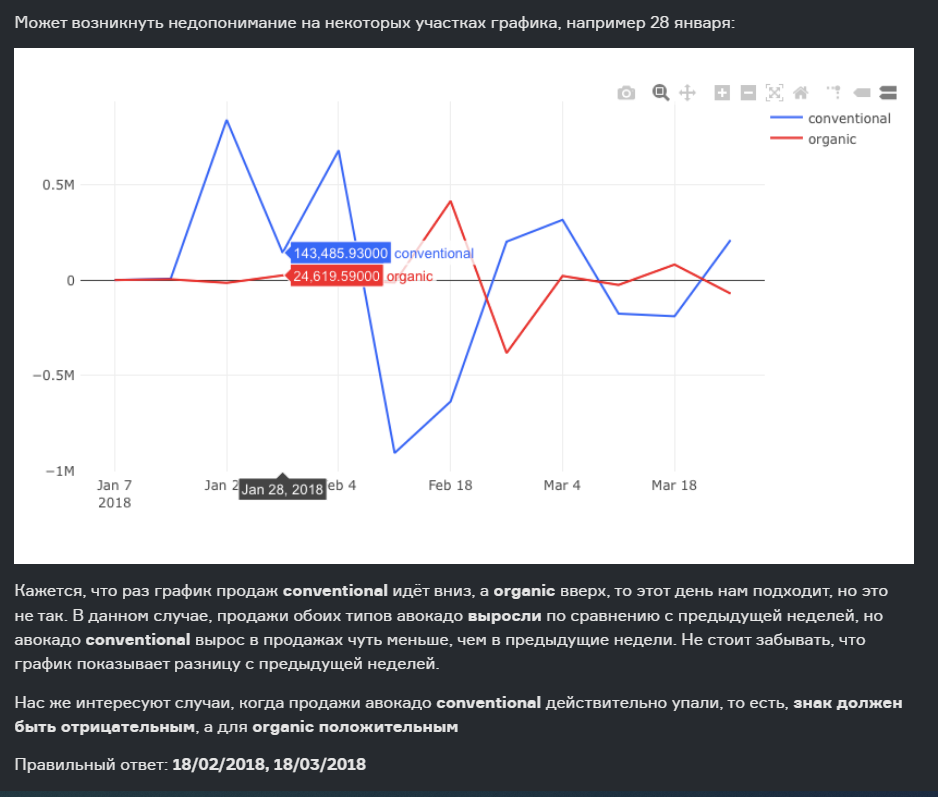
)

ORDER BY

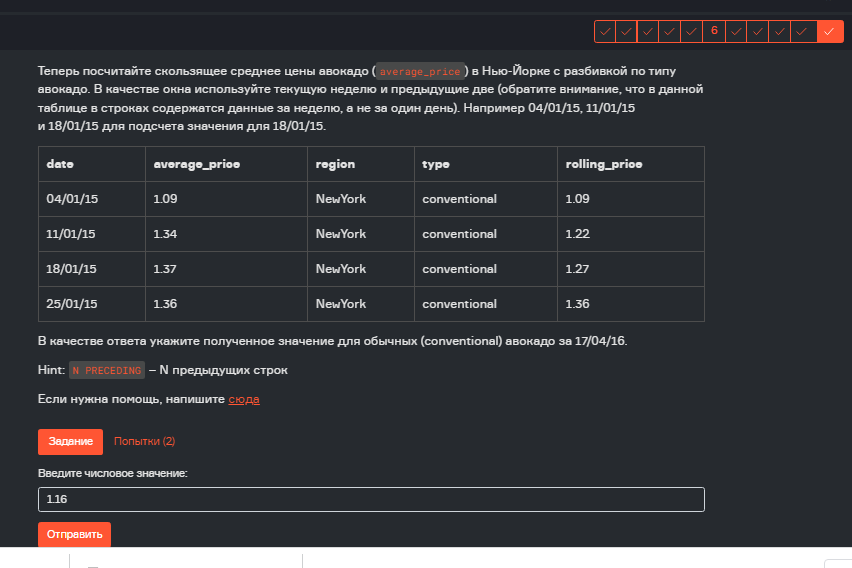
type,

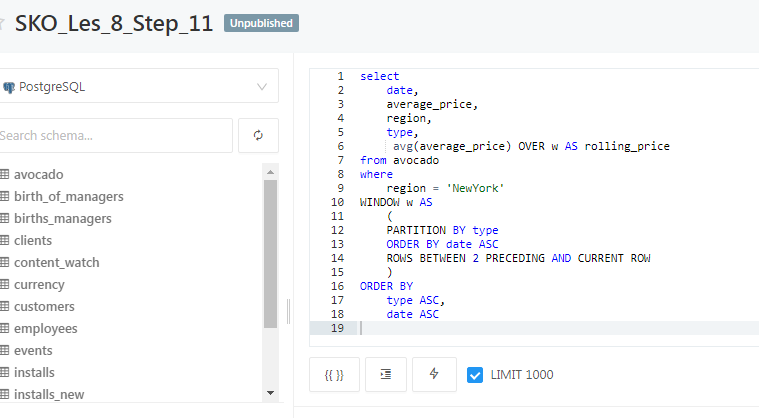
date





11.





Разбор:

Для выполнения этого задания нам будет необходимо увеличить рамку окна и смотреть уже на 2 предыдущие строчки:

WINDOW w AS (

PARTITION BY type

ORDER BY date ASC

ROWS BETWEEN 2 PRECEDING AND CURRENT ROW

)

Нам необходимо посчитать скользящее среднее, поэтому мы будем брать среднее от среднего текущей строчки и двух предыдущих средних.

Пример корректного запроса:

SELECT

type,

date,

average\_price,

avg(average\_price) over w as rolling\_price

FROM

avocado

WHERE

region = 'NewYork'

WINDOW w AS (

PARTITION BY type

ORDER BY date ASC

ROWS BETWEEN 2 PRECEDING AND CURRENT ROW

)

ORDER BY

type,

date

Нам остаётся только посмотреть на скользящее среднее для нужной даты, это можно сделать многими способами, но мы ограничимся уже привычной визуализацией:

