Odd and Even Measurements (Google)

QUESTION

Assume you're given a table with measurement values obtained from a Google sensor over multiple days with measurements taken multiple times within each day.

Write a query to calculate the sum of odd-numbered and even-numbered measurements separately for a particular day and display the results in two different columns. Refer to the Example Output below for the desired format.

Definition:

- Within a day, measurements taken at 1st, 3rd, and 5th times are considered odd-numbered measurements, and measurements taken at 2nd, 4th, and 6th times are considered even-numbered measurements.

Effective April 15th, 2023, the question and solution for this question have been revised.

`measurements` Table:

Column Name	Type
measurement_id	integer

measurement_value	decimal
measurement_time	datetime

`measurements` Example Input:

measurement_id	measurement_value	measurement_time
131233	1109.51	07/10/2022 09:00:00
135211	1662.74	07/10/2022 11:00:00
523542	1246.24	07/10/2022 13:15:00
143562	1124.50	07/11/2022 15:00:00
346462	1234.14	07/11/2022 16:45:00

Step 1: Identify the problem of the case

To create a query that sums odd and even numbered measurements for a certain date and displays the results in two separate columns. We need to know the total of odd and even integers. According to the definition from case, The first, third, and fifth measurements of the day are regarded odd, whereas the second, fourth, and sixth measurements are considered even. The final result will look like this:

Column Name	Type
measurement_day	datetime
odd_sum	decimal
even_sum	decimal

Step 2 : Analyze and solve problems

Using the RANK() windowing function, we first sort the measurements by measurement time and then split them by date. This assists us in determining the sequence of measurements for each day. and also modify the `measurement_time` value to the current date using the command `::DATE` :

```
SELECT *
      , measurement_time::DATE measurement_day
      , RANK() OVER(PARTITION BY measurement_time:: DATE ORDER BY measurement_time)
FROM measurements
```

We may use the following two approaches to filter for odd and even numbers:

- *(`%`) modulus operator: Use `measurement_num% 2!= 0` to see if the result is one, indicating odd numbers, or `measurement_num% 2 = 0` with a result of one, showing even numbers.*
- *`MOD()`: For odd results, use `MOD(measurement_num, 2)!= 0` and for even outcomes, use `MOD(measurement_num, 2) = 0.`*

Finally, we can utilize the modulus notion in conjunction with the `FILTER` clause to sum over the matching `measurement_value` using the aggregate function `SUM()`.

```
SELECT measurement_day
      , SUM(measurement_value) FILTER (WHERE order_value%2 != 0) odd_sum
      , SUM(measurement_value) FILTER (WHERE order_value%2 = 0) even_sum
FROM (
SELECT *
      , measurement_time::DATE measurement_day
      , RANK() OVER(
          PARTITION BY measurement_time:: DATE
          ORDER BY measurement_time
        ) order_value
FROM measurements
) SUB
GROUP BY measurement_day
```

We have the final output:

measurement_day	odd_sum	even_sum
07/10/2022 00:00:00	2355.75	1662.74
07/11/2022 00:00:00	2377.12	2480.70
07/12/2022 00:00:00	2903.40	1244.30