## **1. Window Function SQL**

Now that we can create and query an SQL table, we will now learn about window functions.

## **2. What is a Window Function SQL?**

What is Window Function SQL and how is it useful? SQL window functions express certain very useful operations more simply than regular dataframe dot-notation or regular SQL queries. When processing some rows, each row can use the values of other rows in calculating its value.

## **3. A train schedule**

Suppose we have a table containing a train schedule for a train line. We could use a window function to calculate the time until the next stop and add that as a new column, like so:

## **4. Column with time until next stop added**

A Window function operates on a set of rows and returns a value for each row in the set – but now this value can depend on other rows in the set. The term window describes the set of rows on which the function operates. The value returned for each row can be a value from one of the rows in the “window”, or, a value from a “window function” that uses values from the rows in the window to calculate its value. Let's simplify this example to demonstrate what this means.

## **5. Column with time of next stop**

Here, a window function sql query looked at the current row and the next row, adding a column giving the value of the time column for the following row. Note that in the last row, the value of the new column is empty – that is because there is no following row. Let's look at some code for achieving this result.

## **6. OVER clause and ORDER BY clause**

Now we will see how a window function in this query is able to access more than just the current row, using a specific example. Take a look at this query. This query puts each column on a separate line to make the one that uses a window function more clear. Note the column having the OVER clause – adding an OVER clause designates this query as a window function query. The over clause must contain an ORDER BY clause that tells it how to sequence the rows. The LEAD function lets you query more than one row in a table at a time without having to join the table to itself. In this case, it returns the value of the time column from the next row in the table. Notice how the query constrains the table to only look at rows where train\_id=324. We will now remove that constraint.

## **7. PARTITION BY clause**

This query removes the constraint on train\_id that was in the previous query, and adds a PARTITION BY clause inside the OVER clause. What will the result be? Let's find out.

## **8. Result of adding PARTITION BY clause**

Once we have the time of the current row and the next row together within the same row, it is straightforward for standard sql to calculate the difference.