Advanced SQL: 2nd lesson – Analytic Functions

In the Intro to SQL micro-course, you learned how to use aggregate functions, which perform calculations based on sets of rows. In this tutorial, you'll learn how to define analytic functions, which also operate on a set of rows. However, unlike aggregate functions, analytic functions return a (potentially different) value for each row in the original table.

Analytic functions allow us to perform complex calculations with relatively straightforward syntax. For instance, we can quickly calculate moving averages and running totals, among other quantities.

Syntax:

To understand how to write analytic functions, we'll work with a small table containing data from two different people who are training for a race. The id column identifies each runner, the date column holds the day of the training session, and time shows the time (in minutes) that the runner dedicated to training. Say we'd like to calculate a moving average of the training times for each runner, where we always take the average of the current and previous training sessions. We can do this with the following query:

query = “””

SELECT \*,

AVG(time) OVER(

PARTITION BY id

ORDER BY date

ROWS BETWEEN 1 PRECEDING AND CURRENT ROW

) AS avg\_time

FROM `bigquery-public-data.runners.train\_time`

--- -------------- ------

id date time

--- -------------- ------

1 2019-07-05 22

1 2019-04-15 26

2 2019-02-06 28

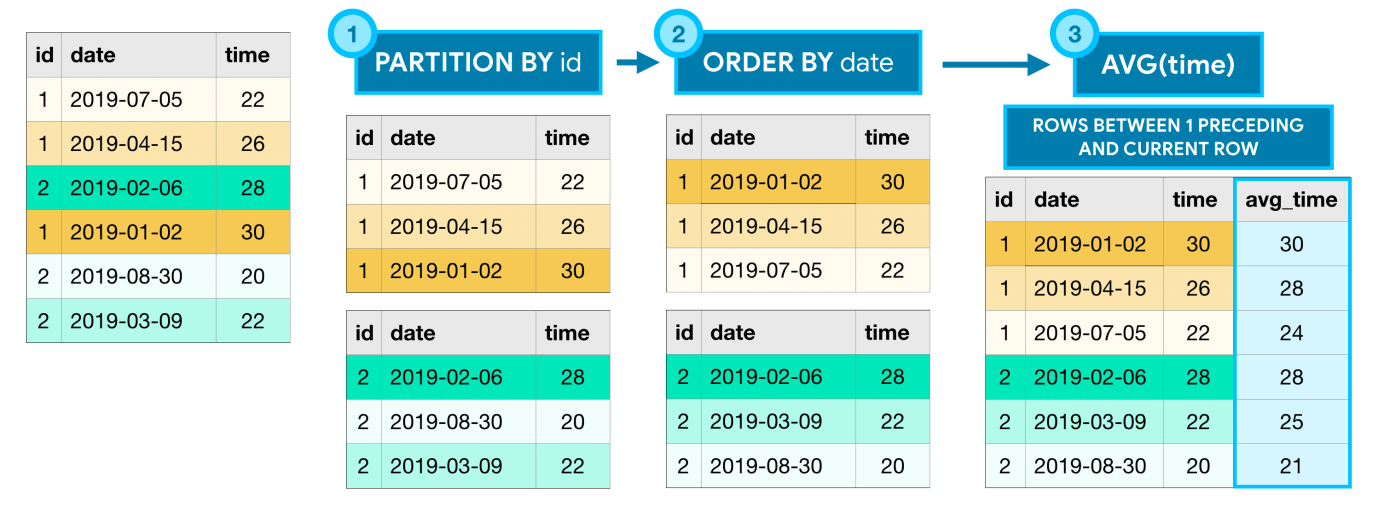
1 2019-01-02 30

2 2019-08-30 20

2 2019-03-09 22

All analytic functions have an OVER clause, which defines the sets of rows used in each calculation. The OVER clause has 3 optional parts:

* The PARTITION BY clause divides the rows of the table into different groups. In the query above, we divide by id so that the calculations are separated by runner.
* The ORDER BY clause defines an ordering within each partition. In the sample query, ordering by the date column ensures that earlier training sessions appear first.
* The final clause (ROWS BETWEEN 1 PRECEDING AND CURRENT ROW) is known as a window frame clause. It identifies the set of rows used in each calculation. We can refer to this group of rows as a window (Actually, analytic functions are sometimes referred to as analytic window functions or simply window functions!).



Advanced window frame clauses:

There are many ways to write window frame clauses.

* ROWS BETWEEN 1 PRECEDING AND CURRENT ROW

The previous row and the current row

* ROWS BETWEEN 3 PRECEDING AND 1 FOLLOWING

The 3 previous rows, the current row, and the following row

* ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING

All rows in the partition

Of course, this is not an exhaustive list, and you can imagine that there are many more options! In the code below, you'll see some of these clauses in action.

3 types of analytic functions:

The example above uses only one of many analytic functions. BigQuery supports a wide variety of analytic functions, and we'll explore a few here.

* Analytic aggregate functions

As you might recall, AVG() (from the example above) is an aggregate function. The OVER clause is what ensures that it's treated as an analytic (aggregate) function. Aggregate functions take all of the values within the window as input and return a single value.

* MIN() or MAX(): returns the minimum or maximum of input values
* AVG() or SUM(): returns the average or sum of input values
* COUNT(): returns the number of rows in the input
* Analytic navigation functions

Navigation functions assign a value based on the value in a (usually) different row than the current row.

* FIRST\_VALUE() or LAST\_VALUE(): returns the first or last value in the input
* LEAD() and LAG(): returns the value on a subsequent or preceding row
* Analytic numbering functions

Numbering functions assign integer values to each row based on the ordering.

* ROW\_NUMBER(): returns the order in which rows appear in the input starting with 1
* RANK(): all rows with the same value in the ordering column receive the same rank value, where the next row receives a rank value which increments by the number of rows with the previous rank value