# Window Functions and More

The only new topic in this file is window functions. By itself, this topic is not difficult, and so we will start out slow. However, this topic is very useful in more complex queries. Solutions to some of these problems will be even longer and more involved than in the previous file. We'll be mixing a lot of prior material

## Window Functions

78) Create a daily cumulative sum of the trading volume of the Yum! stock in `yum`.

SELECT date, SUM(volume)

OVER(ORDER BY date ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS volume

FROM yum

ORDER BY date;

79) Create a cumulative sum of the trading volume of Yum! across months. That is, the final row of this query should be the cumulative sum of all months from 2015 through 2019.

SELECT month, year, SUM(tot\_volume) OVER(ORDER BY year ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS volume

FROM yum\_by\_month;

80) For March 2017, create a table from `yum` with the following columns:

\* Day of the month

\* Row number (ie, the nth trading day of the month)

\* Cumulative low (ie, lowest low so far this month)

\* Cumulative high (ie, highest high so far this month)

\* Cumulative total volume

SELECT date, EXTRACT(day from date) AS day,

COUNT(EXTRACT(day from date)) OVER(ORDER BY date ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS row\_number,

MIN(low) OVER(ORDER BY date ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS minimum,

MAX(high) OVER(ORDER BY date ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS maximum,

SUM(volume) OVER(ORDER BY date ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS volume

FROM yum

WHERE EXTRACT(month from date) = 03

AND EXTRACT(year from date) = 2017

GROUP BY EXTRACT(month from date), date, low, high, volume;

81) From `yum`, create a column that represents the 7-day moving average of the closing price. This syntax is very confusing. You can read more about it [here](<https://www.sqlitetutorial.net/sqlite-window-functions/sqlite-window-frame/>).

SELECT date,

AVG(close) OVER(ORDER BY date ROWS BETWEEN 8 PRECEDING AND 1 PRECEDING) AS moving\_average

FROM yum

GROUP BY close, date;

82) Repeat the March 2017 problem but instead of cumulative highs, lows, and totals, show the 5-day moving highs and lows. (No need for volume here.)

SELECT date, EXTRACT(day from date) AS day,

COUNT(EXTRACT(day from date)) OVER(ORDER BY date ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS row\_number,

MIN(low) OVER(ORDER BY date ROWS BETWEEN 5 PRECEDING AND CURRENT ROW) AS minimum,

MAX(high) OVER(ORDER BY date ROWS BETWEEN 5 PRECEDING AND CURRENT ROW) AS maximum

FROM yum

WHERE EXTRACT(month from date) = 03

AND EXTRACT(year from date) = 2017

GROUP BY EXTRACT(month from date), date, low, high;

83) The [\*\*Williams %R\*\*](https://www.investopedia.com/terms/w/williamsr.asp) is an economic trendline indicator of a stock. Query `yum` to only include two columns: the `date` (unmodified), and the 7-day Williams %R of the stock at that date, call it `williams\_r`. It is computed as follows:

\* Let `h7` be the running 7-day high (ie, highest high of the last 7 trading days).

\* Let `l7` be the running 7-day low (ie, lowest low of the last 7 trading days).

\* `williams\_r = (h7 - close) / (h7 - l7)`

The easiest way to do this problem is to make a CTE containing `h7` and `l7`, and then produce `williams\_r` by querying your CTE.

WITH william (date, wil\_high, wil\_low) AS

(SELECT date, MAX(high) OVER(ORDER BY date ROWS BETWEEN 7 PRECEDING AND 1 PRECEDING) AS h7,

MIN(low) OVER(ORDER BY date ROWS BETWEEN 7 PRECEDING AND 1 PRECEDING) AS l7

FROM yum

)

SELECT y.date, (w.wil\_high - y.close)/(w.wil\_high - w.wil\_low) AS williams\_r

FROM yum y

JOIN william w

ON y.date = w.date;

84) Next, let's create the [\*\*Stochastic Oscillator\*\*](https://www.investopedia.com/terms/s/stochasticoscillator.asp) of `yum`. The stochastic oscillator is actually two lines: One called \_%K\_ and the other called \_%D\_. They are computed as follows:

\* Let `h14` and `l14` denote the 14-day highs and lows (similar to last problem)

\* `percent\_k = (close - l14) / (h14 - l14)`

\* `percent\_d` is the 3-day moving average of `percent\_k`

My solution to this problem involved making \_TWO\_ CTEs (the second one uses the first one). My query was 29 lines long. As a guide to see if you got the answer correct, here's the first few lines of my (sorted) solution:

```

date percent\_k perecent\_d

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

2015-01-02

2015-01-05 -0.7417199

2015-01-06 -0.3257592 -0.7417199

```

WITH p\_k (date, percent\_k) AS

(SELECT date, (close - (MIN(low) OVER(ORDER BY date ROWS BETWEEN 14 PRECEDING AND 1 PRECEDING))) /

((MAX(high) OVER(ORDER BY date ROWS BETWEEN 14 PRECEDING AND 1 PRECEDING)) -

MIN(low) OVER(ORDER BY date ROWS BETWEEN 14 PRECEDING AND 1 PRECEDING))

FROM yum),

p\_d (date, percent\_d) AS

(SELECT y.date, AVG(k.percent\_k) OVER(ORDER BY y.date ROWS BETWEEN 3 PRECEDING AND 1 PRECEDING)

FROM yum y

JOIN p\_k k

ON y.date = k.date)

SELECT y.date, k.percent\_k, d.percent\_d

FROM yum y

JOIN p\_d d

ON y.date = d.date

JOIN p\_k k

ON y.date = k.date;

85) In my opinion, this is the hardest problem in the ladder challenge. For each month between 2015 and 2019, as in the final problem from the `03` file, we'll attach Yum! stock data to the `transactions` data. Let's condense our `yum` data to show relevant monthly statistics. That is, for each month of each year, create a table with the following columns:

\* Year

\* Month

\* The total revenue from our company in `transactions`

\* The monthly low (ie, the lowest low that month)

\* The monthly high (ie, the highest high that month)

\* The monthly open (ie, the opening value in the first trading day that month)

\* The monthly close (ie, the closing value of the last trading day that month)

\* The total trade volume of Yum! that month

My solution to this problem is 38 lines long. For reference, here are the top 3 rows of the solution:

```

year month company\_revenue yum\_low yum\_high yum\_open yum\_close yum\_volume

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

2015 01 $14,106 49.88 53.87 53.12 53.28 89,074,400

2015 02 $20,739 50.68 59.29 56.95 58.31 98,621,800

2015 03 $21,232 54.92 59.55 58.23 58.81 108,827,60

```

Some hints:

\* I used two CTEs, but you may not need to.

\* You'll need the `FIRST\_VALUE()` and `LAST\_VALUE()` window functions.

\* To find the first in each month, you'll need the `PARTITION BY` statement in those window functions. `PARTITION BY` acts a lot like `GROUP BY`, but for window functions.

WITH revenue AS

(SELECT extract(month from orderdate) as month, extract(year from orderdate) as year, to\_char(SUM(quantity \* unit\_price), 'fm$999,999,999') as company\_revenue

FROM transactions

GROUP BY extract(month from orderdate), extract(year from orderdate)

ORDER BY extract(month from orderdate), extract(year from orderdate)

),

yum\_calc AS

(SELECT date, extract(year from date) as year,

FIRST\_VALUE(open) OVER (PARTITION BY EXTRACT(month from date), extract(month from date)) AS yum\_open,

LAST\_VALUE(close) OVER(PARTITION BY EXTRACT(month from date), extract(year from date)) AS yum\_close

FROM yum

),

yum\_info AS

(SELECT EXTRACT(year from date) AS year, EXTRACT(month from date) AS month, MIN(low) AS yum\_low, MAX(high) AS yum\_high, SUM(volume) AS yum\_volume

FROM yum

GROUP BY extract(month from date), extract(year from date)

)

SELECT DISTINCT(EXTRACT(year from y.date)) as year, EXTRACT(month from y.date) as month, r.company\_revenue, yi.yum\_low, yi.yum\_high, yc.yum\_open, yc.yum\_close, yi.yum\_volume

FROM yum y

JOIN revenue r

ON extract(month from y.date) = r.month AND r.year = extract(year from y.date)

JOIN yum\_calc yc

ON yc.date = y.date

JOIN yum\_info yi

ON yi.year = EXTRACT(year from y.date) AND EXTRACT(month from y.date) = yi.month

ORDER BY EXTRACT(year from y.date), EXTRACT(month from y.date);