Running total

TIME SERIES ANALYSIS IN POSTGRESQL

SQL

Jasmin Ludolf
Content Developer, DataCamp

datacamp

What is a running total?

- **Running total:**
  - Total of a sequence of values
  - Cumulative sum
  - Add a new value to the total of all previous values
- Calculate the running total with `SUM() OVER`

```
|values|running_total|
|------|-------------|
|     1|            1|
|     2|            3|
|     3|            6|
```

Calculate the running total

```sql
SELECT
    id,
    timeseries,
    views,
    SUM(views) OVER(
      PARTITION BY id
      ORDER BY timeseries)
FROM views;
```

```
| id|         timeseries|views|sum|
|---|-------------------|-----|---|
|121|2015-12-27 17:34:16|   23| 23|
|121|2015-12-27 17:54:16|   10| 33|
|121|2015-12-27 18:14:16|    8| 41|
|121|2015-12-27 18:34:16|    4| 45|
|211|2015-12-28 03:37:18|    4|  4|
|211|2015-12-28 03:57:18|    2|  6|
|211|2015-12-28 04:17:18|    2|  8|
|211|2015-12-28 04:37:18|    1|  9|
|223|2016-01-01 05:41:22|    7|  7|
|223|2016-01-01 06:01:22|    5| 12|
...
```

The dataset: a train schedule

```sql
SELECT * FROM train_schedule;
```

```
|train_id|       station|arrival_time|
|--------|--------------|------------|
|     324|San Francisco|    07:59:00|
|     324|22nd Street  |    08:03:00|
|     324|Millbrae     |    08:16:00|
|     324|Hillsdale    |    08:24:00|
|     324|Redwood City |    08:31:00|
|     324|Palo Alto    |    08:37:00|
|     324|San Jose     |    09:05:00|
|     217|Gilroy       |    06:06:00|
|     217|San Martin   |    06:15:00|
...
```

[1] https://www.caltrain.com/schedules/weekdaytimetable.html

Lead function

- `LEAD()` : look at values ahead
- `LEAD(value, offset)`

Using the lead function

```
SELECT
    train_id,
    station,
    arrival_time,
    LEAD(arrival_time, 1) OVER (
        PARTITION BY train_id
        ORDER BY arrival_time) AS next_arrival_time
FROM train_schedule
ORDER BY train_id, arrival_time;
```

Using the lead function

```
|train_id|      station|arrival_time|next_arrival_time|
|--------|-------------|------------|-----------------|
|     217|       Gilroy|    06:06:00|         06:15:00|
|     217|   San Martin|    06:15:00|         06:21:00|
|     217|  Morgan Hill|    06:21:00|         06:36:00|
|     217| Blossom Hill|    06:36:00|         06:42:00|
|     217|      Capitol|    06:42:00|         06:50:00|
|     217|       Tamien|    06:50:00|         06:59:00|
|     217|     San Jose|    06:59:00|                 |
|     324|San Francisco|    07:59:00|         08:03:00|
|     324|  22nd Street|    08:03:00|         08:16:00|
|     324|     Millbrae|    08:16:00|         08:24:00|
...
```

Calculating duration of each stop

```
SELECT
    train_id,
    station,
    arrival_time,
    LEAD(arrival_time, 1) OVER (
        PARTITION BY train_id
        ORDER BY arrival_time) AS next_arrival_time,
    LEAD(arrival_time, 1) OVER (
        PARTITION BY train_id
        ORDER BY arrival_time) - arrival_time AS duration
FROM train_schedule
ORDER BY train_id, arrival_time;
```

Calculating duration of each stop

```
|train_id|       station|arrival_time|next_arrival_time|duration|
|--------|--------------|------------|-----------------|--------|
|     217|        Gilroy|    06:06:00|         06:15:00|00:09:00|
|     217|    San Martin|    06:15:00|         06:21:00|00:06:00|
|     217|   Morgan Hill|    06:21:00|         06:36:00|00:15:00|
|     217|  Blossom Hill|    06:36:00|         06:42:00|00:06:00|
|     217|       Capitol|    06:42:00|         06:50:00|00:08:00|
|     217|        Tamien|    06:50:00|         06:59:00|00:09:00|
|     217|      San Jose|    06:59:00|                 |        |
|     324|San Francisco|    07:59:00|
...
```

- Use `SUM() OVER` on `duration` to calculate the running total

Lag function

- `LAG()` : look at values behind
- `LAG(value, offset)`

```
SELECT
    id,
    year_month,
    m_avg,
    LAG(t_monthly_avg, 1) OVER (
        PARTITION BY station_id
        ORDER BY year_month)
        AS previous_m_avg
FROM temperatures_monthly
ORDER BY station_id, year_month;
```

```
|id|year_month|m_avg|previous_m_avg|
|--|----------|-----|--------------|
| 1|2010-01-01| 13.6|              |
| 1|2010-02-01| 14.8|          13.6|
| 1|2010-03-01| 17.0|          14.8|
| 1|2010-04-01| 20.0|          17.0|
| 1|2010-05-01| 24.8|          20.0|
| 1|2010-06-01| 31.3|          24.8|
...
```

Let's practice!

TIME SERIES ANALYSIS IN POSTGRESQL

Running average

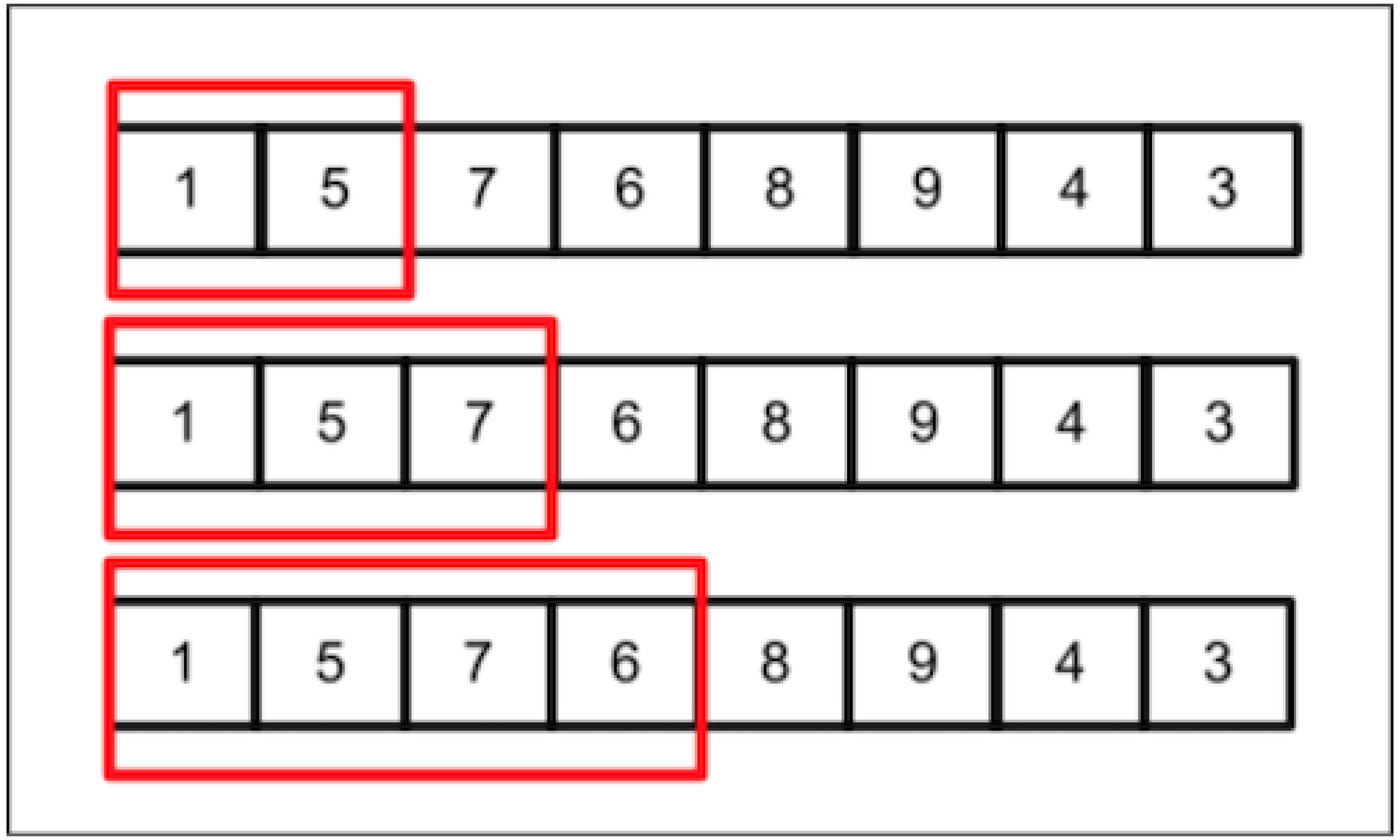
TIME SERIES ANALYSIS IN POSTGRESQL

SQL

Jasmin Ludolf
Content Developer, DataCamp

What is a running average?

- **Running average:**
  - Average of a sequence of values



```
|values|running_average|
|------|---------------|
|     1|              1|
|     5|              3|
|     7|           4.33|
```

Calculate the running average

```sql
SELECT
    id,
    ts,
    views,
    AVG(views) OVER (ORDER BY ts)
      AS running_avg_views
FROM dc_news_fact
WHERE id = '12121'
ORDER BY ts;
```

```
|   id|                  ts|views|running_avg_views|
|-----|-------------------|-----|-----------------|
|12121|2015-12-27 17:14:16| null|             null|
|12121|2015-12-27 17:34:16|   23|23.00000000000000|
|12121|2015-12-27 17:54:16|   10|16.50000000000000|
|12121|2015-12-27 18:14:16|    8|13.66666666666666|
|12121|2015-12-27 18:34:16|    4|11.25000000000000|
|12121|2015-12-27 18:54:16|    7|10.40000000000000|
|12121|2015-12-27 19:14:16|    0|8.666666666666666|
...
```

Running average for multiple time series

```
SELECT
    id,
    ts,
    views,
    AVG(views) OVER (ORDER BY ts)
      AS running_avg_views
FROM dc_news_fact
WHERE id = '12121'
ORDER BY ts;
```

```
SELECT
    id,
    ts,
    views,
    AVG(views) OVER(
      PARTITION BY id ORDER BY ts)
      AS running_avg_views
FROM dc_news_fact
ORDER BY id, ts;
```

Running average over multiple time series: result

```
|   id|                  ts|views|  running_avg_views|
|-----|-------------------|-----|-------------------|
|12121|2015-12-27 17:14:16|     |                   |
|12121|2015-12-27 17:34:16|   23|23.0000000000000000|
...             ...             ...        ...
|12211|2015-12-28 03:17:18|     |                   |
|12211|2015-12-28 03:37:18|    4| 4.0000000000000000|
|12211|2015-12-28 03:57:18|    2| 3.0000000000000000|
|12211|2015-12-28 04:17:18|    2| 2.6666666666666667|
...
```

Let's practice!

TIME SERIES ANALYSIS IN POSTGRESQL

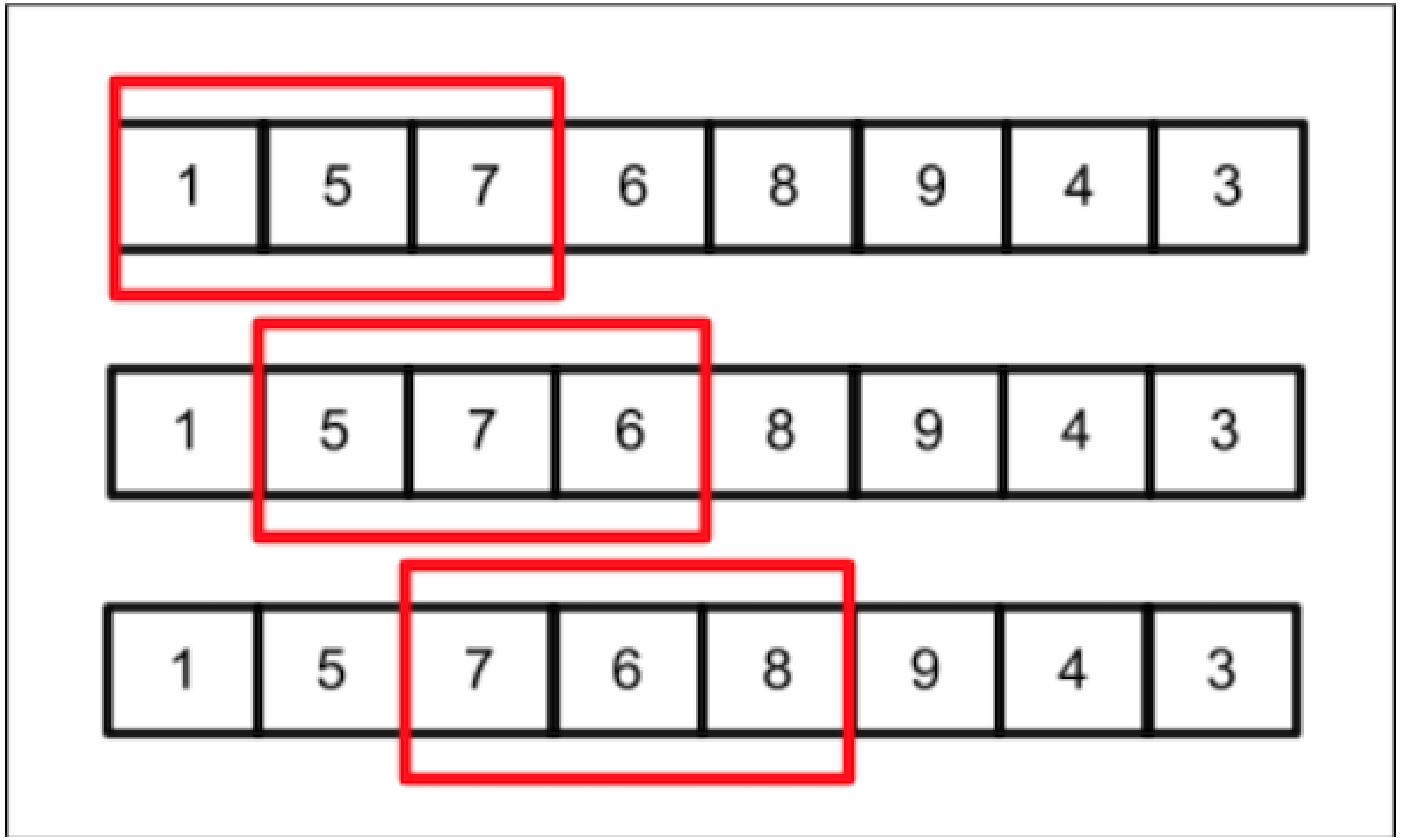
Moving Average

TIME SERIES ANALYSIS IN POSTGRESQL

SQL

Jasmin Ludolf
Content Developer, DataCamp

What is a moving average?



Moving averages

- Specify which rows to average
- Using `ROWS BETWEEN` clause
- `UNBOUNDED PRECENDING AND CURRENT ROW`: look at all previous rows from current row

```
SELECT
id,
views,
AVG(views) OVER(
    PARTITION BY id
    ORDER BY ts
    ROWS BETWEEN
    UNBOUNDED PRECEDING AND CURRENT ROW)
    AS running_avg_views
FROM dc_news_fact;
```

Calculate a moving average

- Calculate the moving average for five rows:
  - `ROWS BETWEEN 4 PRECEDING AND CURRENT ROW`

(1) Row before

(2) Row before

(3) Row before

(4) Row before

(5) Current Row

Calculate a moving average

```
SELECT
    id,
    ts,
    views,
    AVG(views) OVER (
      ORDER BY ts
      ROWS BETWEEN 4 PRECEDING AND CURRENT ROW) AS moving_average_five
FROM dc_news_fact;
```

Calculate a moving average: result

```
|   id|                 ts|views|moving_average_five|
|-----|-------------------|-----|-------------------|
|12121|2015-12-27 17:14:16| null|               null|
|12121|2015-12-27 17:34:16|   23|  23.00000000000000|
|12121|2015-12-27 17:54:16|   10|  16.50000000000000|
|12121|2015-12-27 18:14:16|    8|  13.66666666666666|
|12121|2015-12-27 18:34:16|    4|  11.25000000000000|
|12121|2015-12-27 18:54:16|    7|  10.40000000000000|
|12121|2015-12-27 19:14:16|    0| 5.800000000000000|
...
```

Calculate a moving average without current row

- `ROWS BETWEEN 5 PRECEDING AND 1 PRECEDING`

```sql
SELECT
    stock_symbol,
    day,
    daily_avg,
    AVG(daily_avg) OVER(
        PARTITION BY stock_symbol
        ORDER BY day
        ROWS BETWEEN 5 PRECEDING AND 1 PRECEDING
        ) AS avg_prev_five
FROM daily_stock_averages
ORDER BY stock_symbol, day;
```

Let's practice!

TIME SERIES ANALYSIS IN POSTGRESQL

Wrap-Up

TIME SERIES ANALYSIS IN POSTGRESQL

SQL

Jasmin Ludolf
Content Developer, DataCamp

What we've learned:

- Date and time data types
- How to work with time zones
- How to convert between different date and time data types with `TO_DATE()` and `EXTRACT()` and more
- Manipulate time granularity with `DATE_TRUNC()` and `DATE_PART()`
- How to add, subtract, and aggregate time series data
- Including with statistical aggregates!

What we've learned:

- Partitioning
- Window functions
- Ranking functions
- Running totals
- Running averages
- Moving averages

Congratulations!

TIME SERIES ANALYSIS IN POSTGRESQL