Implementing Row Pattern Recognition

2024/05/29
PGConf.dev at
Vancouver Canada

SRA OSS LLC
Tatsuo Ishii

Who am I?

- Working for SRA OSS LLC as an advisor
- PostgreSQL committer
  - Internationalization of PostgreSQL
  - pgbench, pgstattuple, pgrowlocks, Pgpool-II (an external project)

SRA OSS

Today's talk

- What is Row Pattern Recognition?
- Syntax of Row Pattern Recognition
- Implementation of Row Pattern Recognition and future plans
- Demonstrations

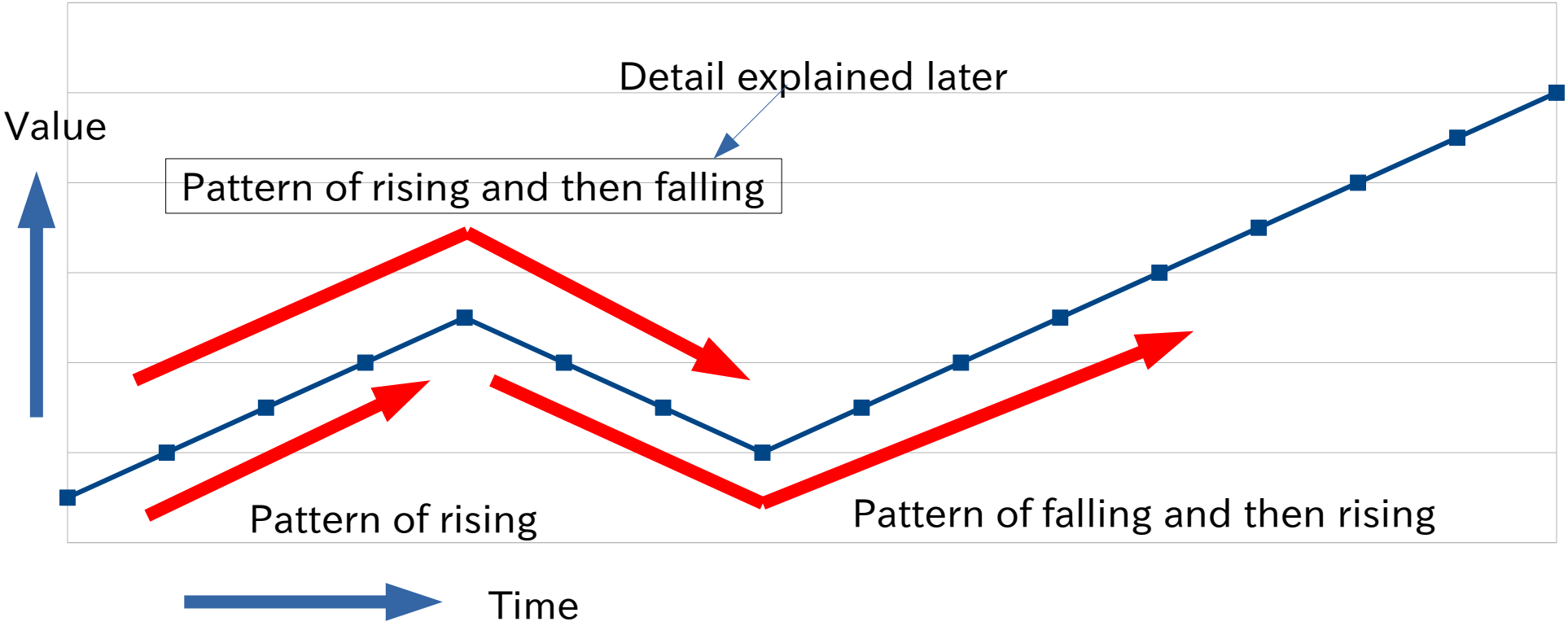
What is Row Pattern Recognition (RPR)?

- One of the features defined by the SQL standard
  - Allow to search for a sequence of rows (e.g. time series data) by “pattern”
- Pattern definition list
  - LOWPRICE AS price < 100
  - UP AS price > PREV(price)
  - DOWN AS price < PREV(price)
- A pattern can be defined by using pattern variables and regular expressions
  - LOWPRICE UP+ DOWN+
- Applications
  - Searching for stock price fluctuation patterns
  - Detecting anomalous values
  - And more

Row Pattern Recognition: definition and implementation

- Relatively new feature first appeared in SQL:2016
  - Called “SQL/RPR” in the standard
- Only Oracle has this at present. No OSS RDBMS has implemented this yet
  - Some Analytics tools implement RPR
    - https://trino.io/

RPR example 1：detecting patterns in a time series data



Search for patterns of rising and then falling

company	tdate	price	
Company1	2023-07-01	100	START
Company1	2023-07-02	200	UP
Company1	2023-07-03	150	DOWN
Company1	2023-07-04	140	DOWN
company1	2023-07-05	150	
Company1	2023-07-06	90	START
Company1	2023-07-07	110	UP
Company1	2023-07-08	130	UP
Company1	2023-07-09	120	DOWN
company1	2023-07-10	130	

Requirement

Find sequences of rows of rising once or more and then falling once or more

Define “row pattern variables”

START AS TRUE,
UP AS price > PREV(price),
DOWN AS price < PREV(price)

Mapped to the first row → START
Mapped to the second or subsequent rows → UP+
Mapped to the third or subsequent rows → DOWN+

Define sequence of patterns using regular expressions

Actual query

```
SELECT company, tdate, price,
first_value(price) OVER w,  -- the first row of the frame
last_value(price) OVER w    -- the last row of the frame
 FROM stock
 WINDOW w AS (
 PARTITION BY company
 ORDER BY tdate
 ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
 AFTER MATCH SKIP PAST LAST ROW
 INITIAL
 PATTERN (START UP+ DOWN+)  -- define patterns
 DEFINE                     -- define pattern variables
  START AS TRUE,
  UP AS price > PREV(price),
  DOWN AS price < PREV(price)
);
```

The result

company	tdate	price	first_value	last_value
company1	2023-07-01	100	100	140
company1	2023-07-02	200		
company1	2023-07-03	150		
company1	2023-07-04	140		
company1	2023-07-05	150		
company1	2023-07-06	90	90	120
company1	2023-07-07	110		
company1	2023-07-08	130		
company1	2023-07-09	120		
company1	2023-07-10	130		

(10 rows)

Example 2：detecting specific consecutive events

alert_time	alert_msg	first_alert	last_alert
2023-11-24 09:00:00	Warning: device is not ready		
2023-11-24 09:01:00	Log: device is busy		
2023-11-24 09:02:00	Log: device is busy		
2023-11-24 09:03:00	Warning: device is not ready		
2023-11-24 09:04:00	Warning: device is not ready		
2023-11-24 09:05:00	Warning: device is not ready		
2023-11-24 09:10:00	Log: device is busy		
2023-11-24 09:20:00	Log: device is busy		
2023-11-24 09:21:00	Warning: device is not ready	2023-11-24 09:21:00	2023-11-24 09:21:20
2023-11-24 09:21:10	Warning: device is not ready		
2023-11-24 09:21:20	Warning: device is not ready		
2023-11-24 09:22:00	Log: device is busy		

Detect events where “Warning” messages appear consecutively within 30 seconds more than 3 times

RPR expression

alert_time	alert_msg
2023-11-24 09:00:00	Warning: device is not ready
2023-11-24 09:01:00	Log: device is busy
2023-11-24 09:02:00	Log: device is busy
2023-11-24 09:03:00	Warning: device is not ready
2023-11-24 09:04:00	Warning: device is not ready
2023-11-24 09:05:00	Warning: device is not ready
2023-11-24 09:10:00	Log: device is busy
2023-11-24 09:20:00	Log: device is busy
2023-11-24 09:21:00	Warning: device is not ready
2023-11-24 09:21:10	Warning: device is not ready
2023-11-24 09:21:20	Warning: device is not ready
2023-11-24 09:22:00	Log: device is busy

Three or more consecutive messages starting with "Warning" appear within 30 seconds more than 3 times

⇩ Define “row pattern variable”

START AS alert_msg LIKE ‘Warning%’,
WARNING AS alert_msg LIKE ‘Warning%’ AND (alert_time – PREV (alert_time) < interval ‘30 seconds’

⇩

START
WARNING
WARNING+

Define sequence of patterns using regular expressions

Actual query

```
SELECT alert_time, alert_msg,
first_value(alert_time) OVER w AS first_alert,
last_value(alert_time) OVER w AS last_alert
 FROM alerts
 WINDOW w AS (
 PARTITION BY device_id
 ORDER BY alert_time
 ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
 INITIAL
 PATTERN (START WARNING WARNING+)
 DEFINE
  START AS alert_msg LIKE 'Warning%',
  WARNING AS alert_msg LIKE 'Warning%' AND
  (alert_time - PREV(alert_time)) < interval '30 seconds'
);
```

The result

alert_time	alert_msg	first_alert	last_alert
2023-11-24 09:00:00	Warning: device is not ready		
2023-11-24 09:01:00	Log: device is busy		
2023-11-24 09:02:00	Log: device is busy		
2023-11-24 09:03:00	Warning: device is not ready		
2023-11-24 09:04:00	Warning: device is not ready		
2023-11-24 09:05:00	Warning: device is not ready		
2023-11-24 09:10:00	Log: device is busy		
2023-11-24 09:20:00	Log: device is busy		
2023-11-24 09:21:00	Warning: device is not ready	2023-11-24 09:21:00	2023-11-24 09:21:20
2023-11-24 09:21:10	Warning: device is not ready		
2023-11-24 09:21:20	Warning: device is not ready		
2023-11-24 09:22:00	Log: device is busy		

Two different types of RPR

- Actually there are two different RPRs
  - R010: Row pattern recognition: FROM clause
    - “MATCH_RECOGNIZE” clause after FROM. RPR is defined there
  - R020: Row pattern recognition: WINDOW clause
    - Define RPR in Window clause
  - R010 and R020 have many common definitions.
- Why I wanted to implement RPR in WINDOW clause?
  - RPR needs to scan a set of tuples over and over again
    - Scanning in FROM clause in the same way is difficult as far as I know
  - WINDOW clause already has such an infrastructure
  - We could be the first implementer of RPR in WINDOW clause in RDBMSs!

Example query in R010

```
SELECT company, tdate, price, m.first_val, m.last_val
FROM stock
MATCH_RECOGNIZE
(
 PARTITION BY company
 ORDER BY tdate
 MEASURES
  FIRST(price) AS first_val,
  LAST(price) AS last_val,
 AFTER MATCH SKIP PAST LAST ROW
 PATTERN (START UP+ DOWN+)
 DEFINE
  START AS TRUE,
  UP AS price > PREV(price),
  DOWN AS price < PREV(price)
) AS m;
```

Syntax of RPR

```
WINDOW window_name AS (
[ PARTITION BY ... ]
[ ORDER BY... ]
[ MEASURES ... ]
ROWS BETWEEN CURRENT ROW AND ...
[ AFTER MATCH SKIP ... ]
[ INITIAL|SEEK ]
PATTERN (...)
[ SUBSET ... ]
DEFINE ...
)
```

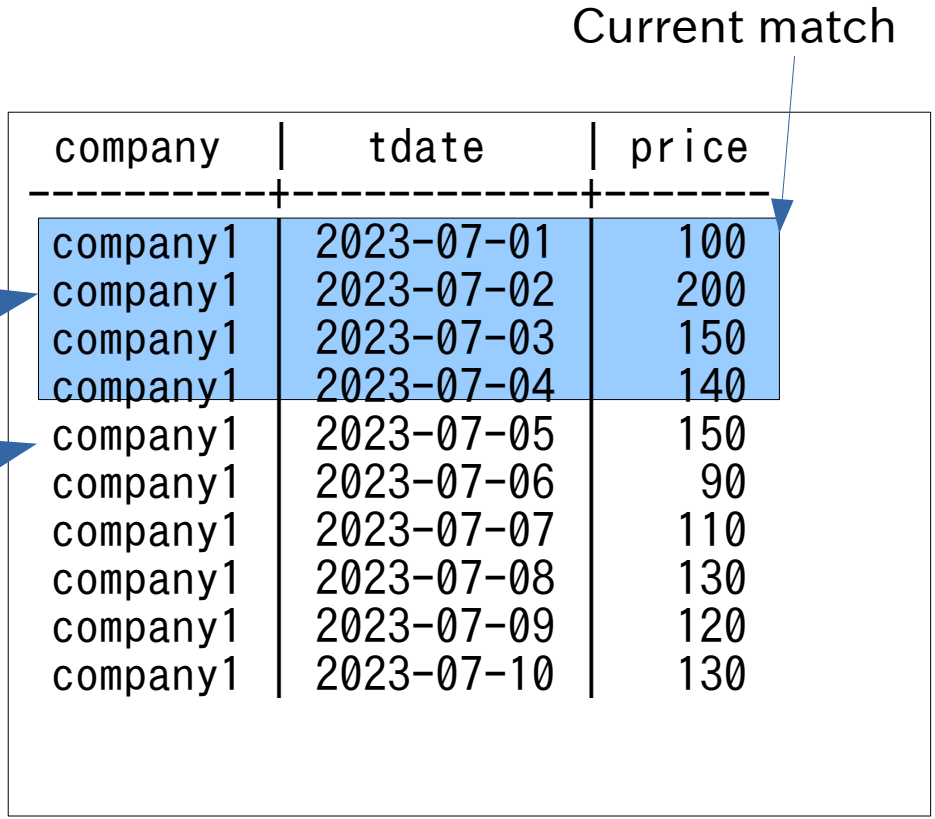
- PARTITION BY, ORDER BY are same as in the Window clause without RPR
- MEASURES and SUBSET are not implemented yet
- Some of sub clauses are not implemented yet

ROWS BETWEEN CURRENT ROW...

- Specify the frame's start and end. Same as Window clause without RPR except only below are allowed in RPR
  - ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
  - ROWS BETWEEN CURRENT ROW AND n FOLLOWING

AFTER MATCH SKIP...

- Only valid with RPR
- Specify where to start the next pattern matching after the pattern matching ends
  - AFTER MATCH SKIP TO NEXT ROW
    - Skip to next row regardless the previous matching rows
  - AFTER MATCH SKIP PAST LAST ROW
    - Skip current matching rows and move to next row
  - AFTER MATCH SKIP TO FIRST|LAST pattern_variable
    - Not supported



INITIAL|SEEK

- Only valid with RPR
- INITIAL
  - Pattern match succeeds only when the set of matching rows starts from the first row of a frame
  - The default
- SEEK
  - Pattern match succeeds even if the set of matching rows do not starts from the first row of a frame
  - Not supported in the current patches

DEFINE (1)

- Only valid with RPR
- Define *pattern_definition _ist*
  - DEFINE variable_name AS search_condition, …
    - search_condition is a boolean logical expression
  - Example pattern definitions
    - START AS TRUE
    - LOWPRICE AS price < 100
    - UP AS price > PREV(price)
    - UP AS price > 100 AND price < PREV(price)

DEFINE (2)

- *Row pattern navigation operations* can be used in DEFINE clause
  - FIRST, LAST, PREV, NEXT
  - Only PREV/NEXT are supported in the current patches
- Following are not supported in the current patch
  - CLASSIFIER (function returning the variable name which matches the pattern)
  - Aggregates
  - Subqueries

PATTERN (1)

- Describe the patterns to be matched using variables in the DEFINE clause
  - `PATTERN (START UP+ DOWN+)`
    - Match with START (the first row)
    - 1 or more rows matching UP follow
    - 1 or more rows matching DOWN follow
  - If a variable A which was not defined in DEFINE clause appears, it is regarded AS “A is TRUE”

PATTERN(2)

- The standard allows following regular expressions with pattern variable
  - +: 1 or more rows
  - *: 0 or more rows
  - ?: 0 or 1 row
  - A | B: OR condition
  - (A B): grouping
  - {n}: n rows
  - {n,}: n or more rows
  - {n,m}: greater or equal to n rows and less than or equal to m rows
  - {,m}: more than 0 and less than or equal to m rows
- The current patch only supports “+” and “*”

Handling aggregates in the target list

- Basically same as aggregates in Window clause without RPR
- Except that the aggregate is applied to only matched rows if RPR clause exists

Example aggregate functions

```
SELECT company, tdate, price,
count(*) OVER w1 AS count1,
count(*) OVER w2 AS count2
 FROM stock
 WINDOW w1 AS (
 PARTITION BY company
 ORDER BY tdate
 ROWS BETWEEN CURRENT ROW AND
 UNBOUNDED FOLLOWING
),
 w2 AS (
 PARTITION BY company
 ORDER BY tdate
 ROWS BETWEEN CURRENT ROW AND
 UNBOUNDED FOLLOWING
 AFTER MATCH SKIP PAST LAST ROW
 INITIAL
 PATTERN (START UP+ DOWN+)
 DEFINE
  START AS TRUE,
  UP AS price > PREV(price),
  DOWN AS price < PREV(price)
);
```

```
 company  |   tdate    | price | count1 | count2
----------+------------+-------+--------+--------
 company1 | 2023-07-01 |   100 |     10 |      4
 company1 | 2023-07-02 |   200 |      9 |      0
 company1 | 2023-07-03 |   150 |      8 |      0
 company1 | 2023-07-04 |   140 |      7 |      0
 company1 | 2023-07-05 |   150 |      6 |      0
 company1 | 2023-07-06 |    90 |      5 |      4
 company1 | 2023-07-07 |   110 |      4 |      0
 company1 | 2023-07-08 |   130 |      3 |      0
 company1 | 2023-07-09 |   120 |      2 |      0
 company1 | 2023-07-10 |   130 |      1 |      0
(10 rows)
```

About RPR Patch

- Implemented in the WINDOW clause
- Proposing patches for PostgreSQL 18 in pgsql-hackers
  - https://www.postgresql.org/message-id/20230625.210509.1276733411677577841.t-ishii%40sranhm.sra.co.jp
  - Patches are for the parser, planner, executor, tests and docs. about 3,000 lines (without tests and documents)
  - New regression test is added to src/test/regress/sql/rpr.sql

Implementation of pattern matching using regular expressions

```
PATTERN (START UP+ DOWN+)
DEFINE
  START AS TRUE,
  UP AS price > PREV(price),
  DOWN AS price < PREV(price)
```

```
price   matched variables
-------------------------------
100     START
200     START UP
150     START DOWN
140     START DOWN
150     START UP
```

Generate possible combinations

```
START START START START START
START UP START START START
START UP DOWN START START
START UP DOWN START START
:
:
START UP DOWN DOWN UP
```

Pattern matching
Using regular expression

START UP+ DOWN+

The pattern match results are expressed as strings. Then do regular expression search against the strings

The longest match is the Final result

START UP DOWN DOWN UP

Future plans

- TODO toward PostgreSQL v18
  - Enhance the pattern matching engine
    - Less memory consumption
- TODO after the first commit
  - Implement more features
    - MEASURE, SUBSET
    - More regular expressions
    - And more...

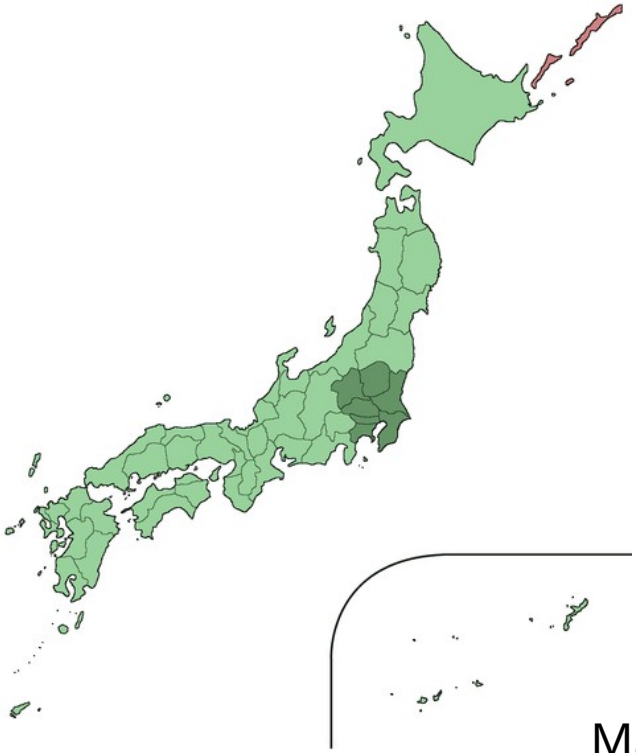
Demonstrations

Search for “extreme hot days”

- What is “extreme hot day”?
  - A day whose maximum temperature exceeds 35 degrees Celsius (95 degrees Fahrenheit)
- What are “extreme hot days”?
  - Consecutive extreme hot days

The data

- Max temperature in a day in major cities around Tokyo, Japan
- Data was downloaded from Japan Meteorological Agency web page



Map: Wikipedia

https://www.data.jma.go.jp/risk/obsdl/index.php

The table

```
CREATE TABLE summer_temperature (
    city text NOT NULL,
    date date NOT NULL,
    highest_tmp numeric(4,2) NOT NULL,
    lowest_tmp numeric(4,2) NOT NULL
);
```

Search for the number of consecutive extreme hot days

Step 1: create a query to find consecutive extreme hot days

```
SELECT city, date, highest_tmp, lowest_tmp,
      first_value(highest_tmp) OVER w,
      last_value(highest_tmp) OVER w,
      max(highest_tmp) OVER w AS high_max,
      count(*) OVER w AS duration,
      avg(highest_tmp) OVER w AS average
FROM summer_temperature
WINDOW w AS (
  PARTITION BY city
  ORDER BY date
  ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
  AFTER MATCH SKIP PAST LAST ROW
  INITIAL
  PATTERN (START M+)
  DEFINE
    START AS TRUE,
    M AS highest_tmp >= 35 AND PREV(highest_tmp) >= 35
);
```

Number of rows matching the pattern “M”

Frames are created for each city and rows are ordered by date

Max temperature in the current row exceeds 35 and the one in the previous row also exceeds 35

Search for consecutive extreme hot days longer than 5 days

Step 2: create a view from the step 1 query

```
CREATE VIEW extreme_hot_days AS
 SELECT city, date, highest_tmp, lowest_tmp,
        first_value(highest_tmp) OVER w,
        last_value(highest_tmp) OVER w,
        max(highest_tmp) OVER w AS high_max,
        count(*) OVER w AS duration,
        avg(highest_tmp) OVER w AS average
 FROM summer_temperature
 WINDOW w AS (
   PARTITION BY city
   ORDER BY date
   ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
   AFTER MATCH SKIP PAST LAST ROW
   INITIAL
   PATTERN (START M+)
   DEFINE
     START AS TRUE,
     M AS highest_tmp >= 35 AND PREV(highest_tmp) >= 35
 );
```

Search for the number of consecutive extreme hot days longer than 5 days

Step 3: query against the view

```
SELECT city, date as start_date, high_max,
duration, trunc(average,2) AS avg FROM extreme_hot_days
WHERE duration::integer >= 5
ORDER BY duration::integer DESC;
    city     | start_date | high_max | duration |  avg
-------------+------------+----------+----------+-------
 Kofu        | 2023-07-23 |    38.70 |        9 | 37.32
 Maebashi    | 2023-07-24 |    38.90 |        8 | 38.05
 Saitama     | 2023-07-24 |    38.80 |        8 | 37.72
 Tokyo       | 2023-07-24 |    37.70 |        8 | 36.43
 Utsunomiya  | 2023-07-25 |    37.00 |        7 | 36.35
 Maebashi    | 2023-08-02 |    38.20 |        6 | 36.70
 Maebashi    | 2023-08-17 |    36.00 |        5 | 35.64
(7 rows)
```

References

- ISO/IEC 19075-5 “Information technology – Guidance for the use of database language SQL – Part5: Row Pattern Recognition”
- Specification of Row Pattern Recognition in the SQL Standard and its Implementations
  - https://link.springer.com/article/10.1007/s13222-022-00404-3
- Trino 426 Documentation (MATCH_RECOGNIZE)
  - https://trino.io/docs/current/sql/match-recognize.html
- Trino 426 Documentation (Row pattern recognition in window structures)
  - https://trino.io/docs/current/sql/pattern-recognition-in-window.htm