

# Pattern Matching with MATCH\_RECOGNIZE

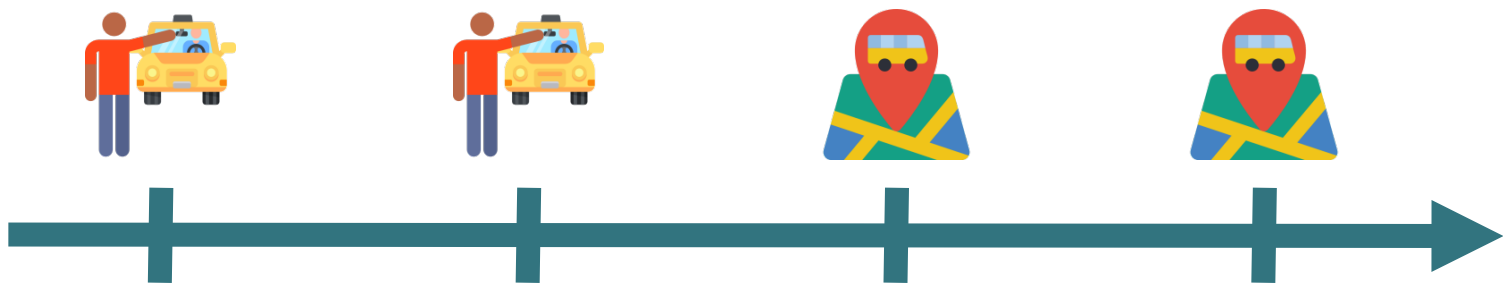
---

## Flink SQL Training

<https://github.com/ververica/sql-training>

# How can I detect Patterns in SQL?

- Find taxi rides with mid-stops



# MATCH\_RECOGNIZE

SQL:2016

# Row Pattern Recognition in SQL

- Flink comes with a complex event processing (CEP) library
- MATCH\_RECOGNIZE clause allows to consolidate CEP and SQL API
- Initial support added in Flink 1.7, more important features in 1.8
- Can be applied to append-only tables that define a time attribute



# Common Use-Cases

- Stock market analysis
- Customer behavior
- Tracking money laundering
- Service quality
- Network intrusion detection
- ...



# Position in a SQL Query

SELECT ...

FROM ...

**MATCH\_RECOGNIZE (...)**

WHERE ...

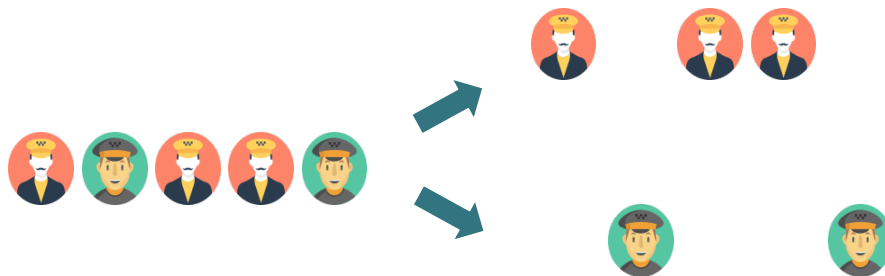
GROUP BY ...



# Example: Rides with Mid-Stops

```
SELECT *  
FROM Rides  
MATCH_RECOGNIZE (  
  PARTITION BY taxiId  
  ORDER BY rideTime  
  MEASURES  
    S.rideId as sRideId  
  AFTER MATCH SKIP PAST LAST ROW  
  PATTERN (S E)  
  DEFINE  
    S AS S.isStart = true,  
    E AS E.isStart = true  
)
```

partition the data by given field  
similar to GROUP BY or keyBy()

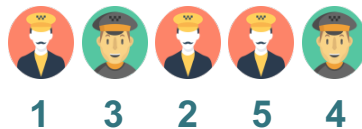


# Example: Rides with Mid-Stops

```
SELECT *  
FROM Rides  
MATCH_RECOGNIZE (  
  PARTITION BY taxiId  
  ORDER BY rideTime  
  MEASURES  
    S.rideId as sRideId  
  AFTER MATCH SKIP PAST LAST ROW  
  PATTERN (S E)  
  DEFINE  
    S AS S.isStart = true,  
    E AS E.isStart = true  
)
```

specify order

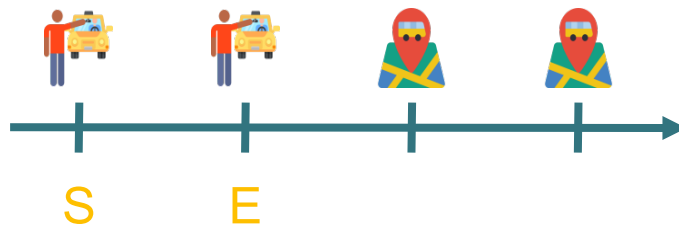
primary order = Event or Processing time





# Example: Rides with Mid-Stops

```
SELECT *  
FROM Rides  
MATCH_RECOGNIZE (  
  PARTITION BY taxiId  
  ORDER BY rideTime  
  MEASURES  
    S.rideId as s RideId  
  AFTER MATCH SKIP PAST LAST ROW  
  PATTERN (S E)  
  DEFINE  
    S AS S.isStart = true,  
    E AS E.isStart = true  
)
```



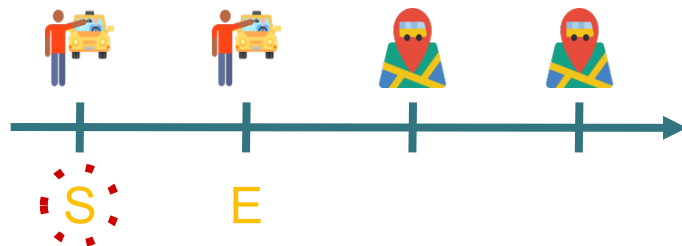
construct a pattern  
with a regular expression-like syntax

# Example: Rides with Mid-Stops

```
SELECT *  
FROM Rides  
MATCH_RECOGNIZE (  
  PARTITION BY taxiId  
  ORDER BY rideTime  
  MEASURES  
    S.rideId as s RideId  
  AFTER MATCH SKIP PAST LAST ROW  
  PATTERN (S E)  
  DEFINE  
    S AS S.isStart = true,  
    E AS E.isStart = true  
)
```

extract measures from matched sequence

defines output of the clause; similar to a SELECT clause



# MATCH\_RECOGNIZE in detail

```
MATCH_RECOGNIZE (  
    PARTITION BY ...  
  
    ORDER BY ...  
  
    MEASURES ...  
  
    ONE ROW PER MATCH  
  
    AFTER MATCH SKIP ...  
  
    PATTERN ...  
  
    DEFINE ...  
)
```



# PATTERN: Defining a Pattern

- Constructed from basic building blocks, called *pattern variables*

PATTERN (A B+ C\* D)

- Operators (quantifiers and other modifiers) can be applied
- DEFINE assigns a meaning to pattern variables



# PATTERN: Defining a Pattern

- *Concatenation:*
  - All rows of a pattern must be mapped to pattern variables
  - A pattern like (A B) means that the contiguity is strict between A and B
  - In other words: No rows between A and B
- *Quantifiers*
  - Number of rows mapped to a pattern variable

*	0 or more rows
+	1 or more rows
?	0 or 1 rows
{ n }, { n, }, { n, m }, { , m }	Define intervals (inclusive)
B*?	Perform mapping <i>reluctant</i> instead of <i>greedy</i> (default behavior)



# DEFINE/MEASURES: Define/Access Variables

- MEASURES

- Defines what will be included in the output of a matching pattern
- Project columns and define expressions for evaluation
- Number of produced rows depends on the output mode.  
Currently, ONE ROW PER MATCH = one output summary row per match only
- Output schema: *[partitioning columns] + [measures columns]*

- DEFINE

- Conditions that a row has to fulfill to be classified to the corresponding variable
- No condition for a variable evaluates to *TRUE*



# DEFINE/MEASURES: Define/Access Variables

- *Pattern Variable Referencing*
  - Access to the set of rows mapped to a particular pattern variable (so far)
  - `A.price` = set of rows mapped so far to A plus the current row if we try to match the current row to A
  - If A is a set, the last row is selected for scalar operations.
  - If no pattern variable is specified (e.g. `SUM(price)`), the default pattern variable "\*" is used. This set contains all rows matched for pattern + current row.

```
PATTERN (A B+)
```

```
DEFINE
```

```
  A AS A.price > 10,
```

```
  B AS B.price > A.price AND
```

```
    SUM(price) < 100 AND SUM(B.price) < 80
```



# DEFINE/MEASURES: Define/Access Variables

- *Pattern Variable Navigation*

- Logical offsets enable navigation within the events that were mapped to a particular pattern variable.
- `FIRST(variable.field, n)`                      n starts from the beginning
- `LAST(variable.field, n)`                      n starts from the end

**PATTERN (A B+)**

**DEFINE**

**A AS A.price > 10,**

**B AS (LAST(B.price, 1) IS NULL OR B.price > LAST(B.price, 1)) AND  
(LAST(B.price, 2) IS NULL OR B.price > 2 \* LAST(B.price, 2))**

- Expressions on same "list" are supported: `LAST(A.price * A.tax)`





# AFTER MATCH SKIP: Continuation strategy

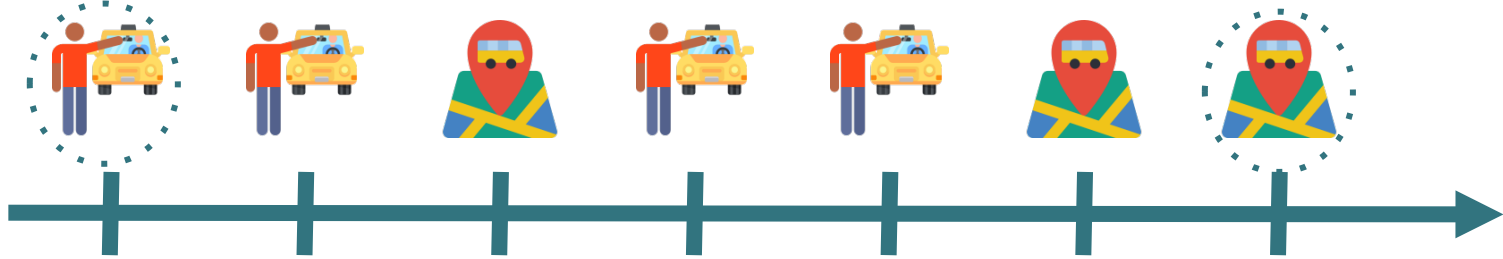
- Location where to start a new matching procedure after a complete match was found

SKIP PAST LAST ROW	next row after the last row of the current match
SKIP TO NEXT ROW	next row after the starting row of the match
SKIP TO LAST variable	last row that is mapped to the specified pattern variable
SKIP TO FIRST variable	first row that is mapped to the specified pattern variable

- Thus, also specifies how many matches a single event can belong to



# Example: Multi-Stop



# Rides with More Than One Mid-Stop

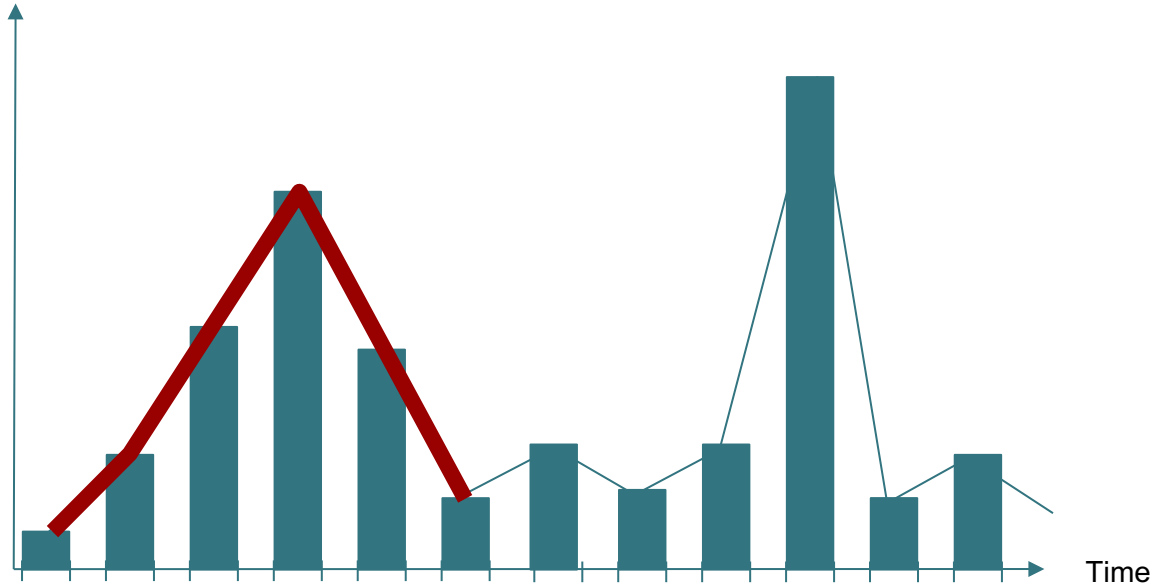
```
SELECT *  
FROM Rides  
MATCH_RECOGNIZE (  
    PARTITION BY taxiId  
    ORDER BY rideTime  
    MEASURES  
        S.rideId as sRideId  
    AFTER MATCH SKIP PAST LAST ROW  
    PATTERN (S E)  
    DEFINE  
        S AS S.isStart = true,  
        E AS E.isStart = true  
)
```

```
SELECT *  
FROM Rides  
MATCH_RECOGNIZE (  
    PARTITION BY taxiId  
    ORDER BY rideTime  
    MEASURES  
        S.rideId as sRideId,  
        COUNT(M.rideId) as countMidStops  
    AFTER MATCH SKIP PAST LAST ROW  
    PATTERN (S M{2,} E)  
    DEFINE  
        S AS S.isStart = true,  
        M AS M.rideId <> S.rideId,  
        E AS E.isStart = false AND  
            E.rideId = S.rideId  
)
```



# Example: Rush (peak) hours - V shape

Number of rides



# Statistics per Area

```
CREATE VIEW RidesInArea AS

SELECT
    toAreaId(lat, lon) as cellId,
    COUNT(distinct rideId) as rideCount,
    TUMBLE_ROWTIME(rideTime, INTERVAL '30' minute) AS rowTime,
    TUMBLE_START(rideTime, INTERVAL '30' minute) AS startTime,
    TUMBLE_END(rideTime, INTERVAL '30' minute) AS endTime

FROM
    Rides

GROUP BY
    toAreaId(lat, lon),
    TUMBLE(rideTime, INTERVAL '30' minute)
```



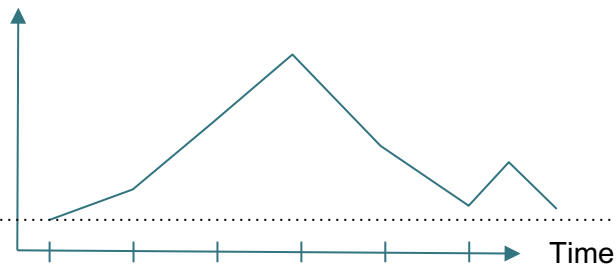
# Rush (peak) hours

```
SELECT * FROM RidesInArea
```

Use previous view.

```
MATCH_RECOGNIZE(  
  PARTITION BY cellId  
  ORDER BY rowTime  
  MEASURES  
    FIRST(UP.startTime) as rushStart,  
    LAST(DOWN.endTime) AS rushEnd,  
    SUM(UP.rideCount) + SUM(DOWN.rideCount) AS rideSum  
  AFTER MATCH SKIP PAST LAST ROW  
  PATTERN (UP{4,} DOWN{2,} E)  
  DEFINE  
    UP AS UP.rideCount > LAST(UP.rideCount, 1) OR  
      LAST(UP.rideCount, 1) IS NULL,  
    DOWN AS DOWN.rideCount < LAST(DOWN.rideCount, 1) OR  
      LAST(DOWN.rideCount, 1) IS NULL,  
    E AS E.rideCount > LAST(DOWN.rideCount)  
)
```

Number of rides



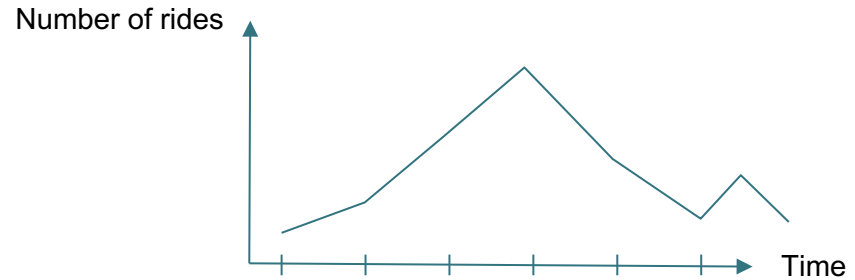
Apply match to result of the inner query.



# Rush (peak) hours

```
SELECT * FROM RidesInArea
```

```
MATCH_RECOGNIZE(  
  PARTITION BY cellId  
  ORDER BY rowTime  
  MEASURES  
    FIRST(UP.startTime) as rushStart,  
    LAST(DOWN.endTime) AS rushEnd,  
    SUM(UP.rideCount) + SUM(DOWN.rideCount) AS rideSum  
  AFTER MATCH SKIP PAST LAST ROW  
  PATTERN (UP{4,} DOWN{2,} E)  
  DEFINE  
    UP AS UP.rideCount > LAST(UP.rideCount, 1) OR  
      LAST(UP.rideCount, 1) IS NULL,  
    DOWN AS DOWN.rideCount < LAST(DOWN.rideCount, 1) OR  
      LAST(DOWN.rideCount, 1) IS NULL,  
    E AS E.rideCount > LAST(DOWN.rideCount)  
)
```



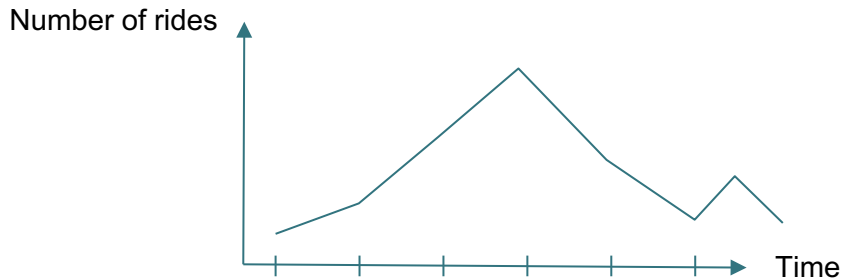
Access elements of looping pattern.



# Rush (peak) hours

```
SELECT * FROM RidesInArea
```

```
MATCH_RECOGNIZE(  
  PARTITION BY cellId  
  ORDER BY rowTime  
  MEASURES  
    FIRST(UP.startTime) as rushStart,  
    LAST(DOWN.endTime) AS rushEnd,  
    SUM(UP.rideCount) + SUM(DOWN.rideCount) AS rideSum  
  AFTER MATCH SKIP PAST LAST ROW  
  PATTERN (UP{4,} DOWN{2,} E)  
  DEFINE  
    UP AS UP.rideCount > LAST(UP.rideCount, 1) OR  
      LAST(UP.rideCount, 1) IS NULL,  
    DOWN AS DOWN.rideCount < LAST(DOWN.rideCount, 1) OR  
      LAST(DOWN.rideCount, 1) IS NULL,  
    E AS E.rideCount > LAST(DOWN.rideCount)  
)
```



Aggregate values from looping patterns





# Feature set of MATCH\_RECOGNIZE

- Quantifiers support
  - +, \*, {x,y} , Greedy (default), ? (reluctant)
  - With some restrictions (not working for last pattern)
- After Match Skip
  - Skip to first/last, skip past last, skip to next
- Aggregates and UDFs (1.8)
- Allow time attribute extraction (1.8)
- Time constraints using the WITHIN clause (1.8)
- Not supported: alter(|), permute, exclude '{- -}'





# ververica

---

[www.ververica.com](http://www.ververica.com)

@VervericaData