Flink CEP SQL详解

1 语法

```
SELECT [ ALL | DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression
[MATCH_RECOGNIZE (
[PARTITION BY {partitionItem [, partitionItem]*}]
[ORDER BY {orderItem [, orderItem]*}]
[MEASURES {measureItem AS col [, measureItem AS col]*}]
[ONE ROW PER MATCH|ALL ROWS PER MATCH|ONE ROW PER MATCH WITH TIMEOUT ROW
S|ALL ROWS PER MATCH WITH TIMEOUT ROWS]
[AFTER MATCH SKIP]
PATTERN (patternVariable[quantifier] [ patternVariable[quantifier]]*) WITH
IN intervalExpression
DEFINE {patternVariable AS patternDefinationExpression [, patternVariable
 AS patternDefinationExpression]*}
)];
```

1.1 参数说明

参数	说明
PARTITION BY	分区的列，可选项。
ORDER BY	可指定多列，但是必须以EVENT TIME列或者PROCESS T
MEASURES	定义如何根据匹配成功的输入事件构
ONE ROW PER MATCH	对于每一次成功的匹配，只会产生一
ONE ROW PER MATCH WITH TIMEOUT ROWS	除了匹配成功的时候产生输出外，超时的时候也会产生输出。超 句定义。
ALL ROWS PER MATCH	对于每一次成功的匹配，对应于每一个输入事件，
ALL ROWS PER MATCH WITH TIMEOUT ROWS	除了匹配成功的时候产生输出外，超时的时候也会产生输出。超 句定义。
[ONE ROW PER MATCH	ALL ROWS PER MATCH
AFTER MATCH SKIP TO NEXT ROW	匹配成功之后，从匹配成功的事件序列中的第一个事件的

AFTER MATCH SKIP PAST LAST ROW	匹配成功之后，从匹配成功的事件序列中的最后一个事件的
AFTER MATCH SKIP TO FIRST patternItem	匹配成功之后，从匹配成功的事件序列中第一个对应于patte
AFTER MATCH SKIP TO LAST patternItem	匹配成功之后，从匹配成功的事件序列中最后一个对应于pat
PATTERN	定义待识别的事件序列需要满足的规则，需要定义在()中，由

说明：

① patternVariable之间如果以空格间隔，表示符合这两种patternVariable的事件中间不存在其他事件。

② patternVariable之间如果以->间隔，表示符合这两种patternVariable的事件之间可以存在其它事件。

1.2 参数解释

（1）quantifier

quantifier用于指定符合patternVariable定义的事件的出现次数。

参数	说明
*	0次或多次
+	1次或多次
?	0次或1次
{n}	n次
{n,}	大于等于n次
{n, m}	大于等于n次，小于等于m次
{,m}	小于等于m次

默认为贪婪匹配。例如，对于pattern: A -> B+ -> C，输入为a bc1 bc2 c（其中bc1和bc2表示既匹配B也匹配C），则输出为a bc1 bc2 c。可以在quantifier符号后面加？来表示非贪婪匹配。例如：

*?

+?

{n}?

{n,}?

{n, m}?

{,m}?

此时，对于上面例子中的PATTERN及输入，产生的输出为a bc1 bc2,a bc1 bc2 c。

WITHIN定义符合规则的事件序列的最大时间跨度。

静态窗口格式：INTERVAL ‘string’ timeUnit [TO timeUnit]。例如：INTERVAL ‘10’

SECOND, INTERVAL ‘45’ DAY, INTERVAL ‘10:20’ MINUTE TO SECOND, INTERVAL ‘10:20.10’ MINUTE TO SECOND, INTERVAL ‘10:20’ HOUR TO MINUTE, INTERVAL ‘1-5’ YEAR TO MONTH。

动态窗口格式：INTERVAL intervalExpression。例如：INTERVAL A.windowTime + 10，其中A为PATTERN定义中第一个patternVariable。在intervalExpression的定义中，可以使用PATTERN定义中出现过的patternVariable。当前只能使用第一个patternVariable。intervalExpression中可以使用UDF，intervalExpression的结果必须为Long，单位为millisecond，表示窗口的大小。

EMIT TIMEOUT 定义多个超时窗口，所有超时窗口不能超过WITHIN语句中定义的窗口，可选项：

① 固定窗口：

格式：EMIT TIMEOUT (INTERVAL ‘string’ timeUnit [TO timeUnit], INTERVAL ‘string’ timeUnit [TO timeUnit], …)

示例：EMIT TIMEOUT (INTERVAL ‘2’ HOUR, INTERVAL ‘6’ HOUR)

② 周期性窗口：

格式：EMIT TIMEOUT EVERY INTERVAL ‘string’ timeUnit [TO timeUnit]

示例：EMIT TIMEOUT EVERY INTERVAL ‘2’ HOUR

EMIT TIMEOUT语句必须与WITHIN语句及ONE ROW PER MATCH WITH TIMEOUT ROWS或者ALL ROW PER MATCH WITH TIMEOUT ROWS同时使用。

EMIT TIMEOUT语句主要用在有多个超时指标需要计算的场景，如物流配送中可能需要同时统计订单超时2小时未配送、超时4小时未配送等。

DEFINE 定义在PATTERN中出现的patternVariable的具体含义，若某个patternVariable在DEFINE中没有定义，则认为对于每一个事件，该patternVariable都成立。

（2）MEASURES和DEFINE语句中函数

函数	函数意义
Row Pattern Column References	形式为：patternVariable.col。表示访问patternVariable所对应的事件
PREV	只能用在DEFINE语句中，通常与Row Pattern Column References合用。用于访问指定的PAT offset所对应的事件的指定的列。例如，DOWN AS DOWN.price < PREV(DOWN.price)，PREV price列的值。说明：① DOWN.price等价于PREV(DOWN.price, 0)。② PREV(DOWN.pric
FIRST、LAST	一般与Row Pattern Column References合用，用于访问指定的PATTERN所对应的事件序列 FIRST(A.price, 3)表示PATTERN A所对应的事件序列中的第4个事件。LAST(A.price, 3)表示PATT 事件。

（3）输出列

函数	输出列

ONE ROW PER MATCH	包括PARTITION BY中指定的列及MEASURES中定义的列。 对于MEASURES中无需重复定义。
ONE ROW PER MATCH WITH TIMEOUT ROWS	除匹配成功的时候产生输出外，超时的时候也会产生输出，超时时义。

说明：① 当定义PATTERN时，最好也定义WITHIN，否则可能会造成STATE越来越大；② ORDER BY中定义的首列必须为EVENT TIME列或者PROCESS TIME列。

2 示例

（1）示例语法

```
SELECT *
FROM Ticker MATCH_RECOGNIZE (
PARTITION BY symbol
ORDER BY tstamp
MEASURES STRT.tstamp AS start_tstamp,
LAST(DOWN.tstamp) AS bottom_tstamp,
LAST(UP.tstamp) AS end_tstamp
ONE ROW PER MATCH
AFTER MATCH SKIP TO NEXT ROW
PATTERN (STRT DOWN+ UP+) WITHIN INTERVAL '10' SECOND
DEFINE
DOWN AS DOWN.price < PREV(DOWN.price),
UP AS UP.price > PREV(UP.price)
);
```

（2）测试数据

timestamp(TIMESTAMP)	card_id(VARCHAR)	location(VARCHAR)
2018-04-13 12:00:00	1	Beijing
2018-04-13 12:05:00	1	Shanghai
2018-04-13 12:10:00	1	Shenzhen
2018-04-13 12:20:00	1	Beijing

（3）测试案例语法

当相同的card_id在十分钟内，从两个不同的location发生刷卡现象，就会触发报警机制，以便于监测信用卡盗刷等现象。

```
CREATE TABLE datahub_stream (
    `timestamp`               TIMESTAMP,
```

```
    card_id                    VARCHAR,
    location                   VARCHAR,
    `action`                   VARCHAR,
    WATERMARK wf FOR `timestamp` AS withOffset(`timestamp`, 1000)
) WITH (
    type = 'datahub'
    ...
);

CREATE TABLE rds_out (
    start_timestamp              TIMESTAMP,
    end_timestamp                TIMESTAMP,
    card_id                      VARCHAR,
    event                        VARCHAR
) WITH (
    type= 'rds'
    ...
);
--定义计算逻辑。
insert into rds_out
select
`start_timestamp`,
`end_timestamp`,
card_id, `event`
from datahub_stream
MATCH_RECOGNIZE (
    PARTITION BY card_id   --按card_id分区，将相同卡号的数据分到同一个计算节点上
。
    ORDER BY `timestamp`   --在窗口内，对事件时间进行排序。
    MEASURES            --定义如何根据匹配成功的输入事件构造输出事件。
        e2.`action` as `event`,
        e1.`timestamp` as `start_timestamp`,   --第一次的事件时间为start_ti
mestamp。
        LAST(e2.`timestamp`) as `end_timestamp` --最新的事件时间为end_times
tamp。
    ONE ROW PER MATCH           --匹配成功输出一条。
    AFTER MATCH SKIP TO NEXT ROW --匹配后跳转到下一行。
    PATTERN (e1 e2+) WITHIN INTERVAL '10' MINUTE  --定义两个事件，e1和e2。
    DEFINE                     --定义在PATTERN中出现的patternVariable的具体
含义。
        e1 as e1.action = 'Consumption',    --事件一的action标记为Consumpti
on。
        e2 as e2.action = 'Consumption' and e2.location <> e1.location --
事件二的action标记为Consumption，且事件一和事件二的location不一致。
);
```

（4）测试结果

start_timestamp(TIMESTAMP)	end_timestamp(TIMESTAMP)	card_id(VAR
2018-04-13 12:00:00.0	2018-04-13 12:05:00.0	1
2018-04-13 12:05:00.0	2018-04-13 12:10:00.0	1