d-one.ai

Apache Flink CEP Cheat Sheet

Apache Flink CEP allows you to detect event patterns in an endless stream of events, giving you the opportunity to get hold of what's important in your data. Each pattern or state has:

- **Name**: The name of pattern / state. This allows you to retrieve elements that match this pattern, similar to groups in a regex.
- **Conditions**: Criteria that define if an incoming event is part of the pattern.

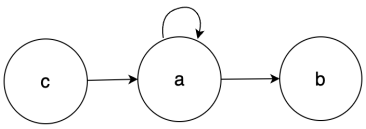
In that regard, Complex Event Processing can be considered as a state machine and its transitions, similar to a regular expression (regex).
You can find the full documentation for Flink CEP here[1].

Example

```
Pattern.<Event>begin("start")
       .where(SimpleCondition.of(value -> value.getName().equals("c")))
       .followedBy("middle")
       .where(SimpleCondition.of(value -> value.getName().equals("a")))
       .oneOrMore()
       .consecutive()
       .followedBy("end1")
       .where(SimpleCondition.of(value -> value.getName().equals("b")));
```

Explanation:

1. The pattern begins with the state named `"start"`.
   Condition: The event's `name` must be `"c"`.
2. `.followedBy("middle")`: Defines the next state `"middle"`.
   Condition: The event's `name` must be `"a"`.
   - `.oneOrMore()`: Requires `"middle"` to match one or more times, allowing consecutive `"a"` events.
   - `.consecutive()`: Enforces that the `"middle"` matches are consecutive, with no other events between them.
3. `.followedBy("end1")`: Defines the final state as `"end1"`.
   Condition: The event's `name` must be `"b"`.



Equivalent Regex: `(ca+b)`

[1] https://nightlies.apache.org/flink/flink-docs-release-1.17/docs/libs/cep/

d-one.ai

The Pattern API Essentials

Here's a quick overview of essential Flink CEP commands. For more details, refer to the official documentation.

`Pattern.begin("state")`
Starts a new pattern with a named state. Use this to define the entry point of a pattern.

`Pattern.where(condition)`
Adds a condition to the current state, filtering events that satisfy it. Use it to ensure only specific events can proceed to the next state.

`Pattern.next("state")`
Specifies the next strict state transition, meaning events must occur in strict sequence. Use this when exact ordering is crucial.

`Pattern.followedBy("state")`
Specifies a loose (non-strict) transition to the next state. Use this when events should appear in order, but not necessarily consecutively.

`Pattern.followedByAny("state")`
Allows transitions to the next state regardless of intermediate events. Use this when only the sequence is important.

`Pattern.times(n)`
Specifies that a state should match exactly n times. Useful for repeated events.

`Pattern.timesOrMore(n)`
Specifies that a state should match at least n times. Useful for repeated events.

`Pattern.optional()`
Marks a state as optional. Use it to allow matching patterns even if this state is missing.

`Pattern.until(condition)`
Defines a condition that, if met, will stop matching further events in that state. Use this to halt on specific criteria.

`Pattern.within(time)`
Sets a time window for the pattern to be matched. Use it to constrain matches to a specific timeframe.

d-one.ai

After Match Skip Strategy

The **After Match Skip Strategy** in Flink CEP defines what happens after a pattern match is found. It controls how the event sequence proceeds after detecting a match. There are three common strategies:

1. **Skip to Next (`SkipStrategy.SKIP_TO_NEXT`)**
   Skips to the next event after a match is found, starting the pattern search from the next event.
2. **Skip Past Last Event (`SkipStrategy.SKIP_PAST_LAST_EVENT`)**
   Skips past the last event of the match, continuing the search for new patterns starting after the matched event.
3. **No Skip (`SkipStrategy.NO_SKIP`)**
   Does not skip any events, and the pattern matching continues from the current position, which may lead to overlapping patterns.

Each strategy helps manage how the stream is processed after a match, allowing for flexible pattern matching and event handling.

Flink SQL

If you are using Flink SQL for the exercises, you will need to use the pattern matching queries. You can see a template below, and more information is available here[2].

```
SELECT T.aid, T.bid, T.cid
FROM MyTable
    MATCH_RECOGNIZE (
      PARTITION BY userid
      ORDER BY proctime
      MEASURES
        A.id AS aid,
        B.id AS bid,
        C.id AS cid
      PATTERN (A B C)
      DEFINE
        A AS name = 'a',
        B AS name = 'b',
        C AS name = 'c'
    ) AS T
```

PATTERN (A B C): Specifies the desired sequence of events to match. Events should occur in the exact order of `A`, followed by `B`, then `C`.
You can describe your pattern using repetition operators like `*` (zero or more), `+` (one or more), `?` (optional), and parentheses for grouping, along with alternation (`|`) and permutations (`PERMUTE`).

[2] https://nightlies.apache.org/flink/flink-docs-release-1.17/docs/dev/table/sql/queries/match_recognize/#pattern-recognition