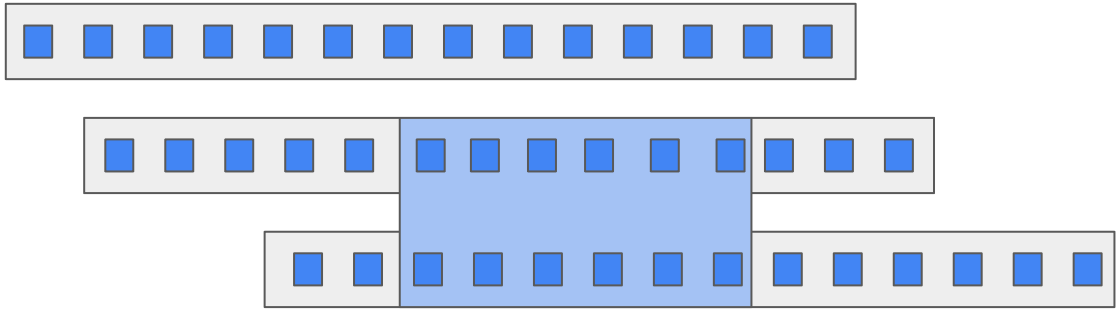
EPL Reference : Patterns

Introduction

- Definition: The Pattern function in EPL is used to search for correspondences between a pattern and a set of model elements.
- Structure:
  a. Pattern: Describes the structure you're looking for. It consists of named and typed roles, and a match condition.
  b. Roles: Represent the various elements of the pattern. They can be classes, attributes, methods, etc.
  c. Match condition: Determines which elements match the pattern. It can be a simple or complex Boolean expression.



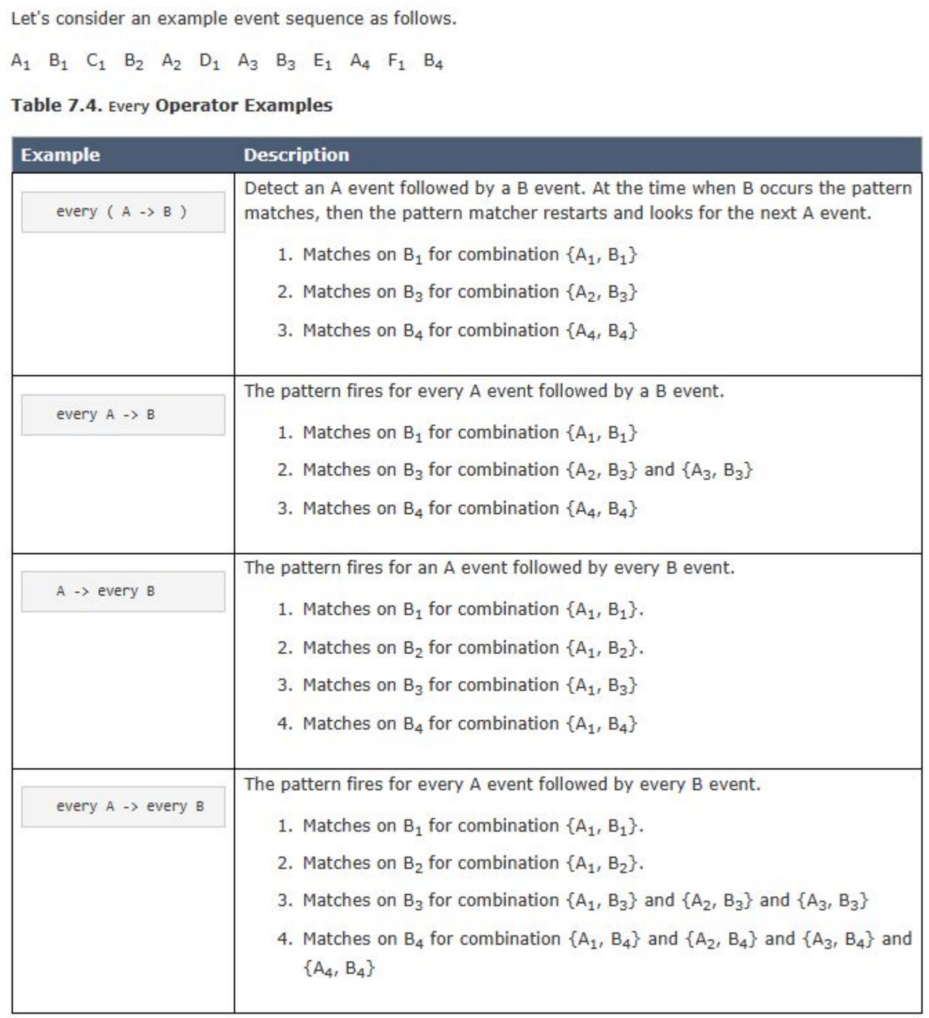
Illustrative example

- This pattern monitors the values of an attribute and triggers an alert if an abnormal value is detected.
- The pattern uses a comparison operator to determine if a value is abnormal.
- The pattern can be configured to define the anomaly threshold and the type of alert to trigger.

Database : G1, N1, P1, L1, P2, G2, G3, N2, L2, P3, N3, L3, P4, G4, N4, L4		
Every N followed by L	Every P followed by G	G followed by N
• N1 - L1 • N2 - L2 • N3 - L3 • N4 - L4	• P1 - G2 • P2 - G3 • P3 - G4	• G1 - N2 • G2 - N3 • G3 - N4

Course's example

REFER TO : Table 7.4. `Every` Operator Examples



Let's consider an example event sequence as follows.

A_1 B_1 C_1 B_2 A_2 D_1 A_3 B_3 E_1 A_4 F_1 B_4

Table 7.4. `Every` Operator Examples

Example	Description
`every ( A -> B )`	Detect an A event followed by a B event. At the time when B occurs the pattern matches, then the pattern matcher restarts and looks for the next A event. 1. Matches on B_1 for combination $\{A_1, B_1\}$ 2. Matches on B_3 for combination $\{A_2, B_3\}$ 3. Matches on B_4 for combination $\{A_4, B_4\}$
`every A -> B`	The pattern fires for every A event followed by a B event. 1. Matches on B_1 for combination $\{A_1, B_1\}$ 2. Matches on B_3 for combination $\{A_2, B_3\}$ and $\{A_3, B_3\}$ 3. Matches on B_4 for combination $\{A_4, B_4\}$
`A -> every B`	The pattern fires for an A event followed by every B event. 1. Matches on B_1 for combination $\{A_1, B_1\}$. 2. Matches on B_2 for combination $\{A_1, B_2\}$. 3. Matches on B_3 for combination $\{A_1, B_3\}$ 4. Matches on B_4 for combination $\{A_1, B_4\}$
`every A -> every B`	The pattern fires for every A event followed by every B event. 1. Matches on B_1 for combination $\{A_1, B_1\}$. 2. Matches on B_2 for combination $\{A_1, B_2\}$. 3. Matches on B_3 for combination $\{A_1, B_3\}$ and $\{A_2, B_3\}$ and $\{A_3, B_3\}$ 4. Matches on B_4 for combination $\{A_1, B_4\}$ and $\{A_2, B_4\}$ and $\{A_3, B_4\}$ and $\{A_4, B_4\}$

Syntax summary

Step 1 : Defining the Pattern Atom

Let’s imagine that we have a data stream containing events called *DoorEvent*. Each *DoorEvent* has an attribute named *type* that specifies the state of the door (open or closed).

We define a pattern atome named *openDoor* that **acts like a filter**. This filter looks for any incoming *DoorEvent* where the type attribute is equal to “open”

Syntax summary

Step 2 : Defining the main Pattern

The main pattern, named *openDoorPattern*, simply consists of the single pattern atom *openDoor* we defined earlier. In other words, the main pattern is looking for any event that matches the criteria specified in *onpenDoor*.

Syntax summary

Step 3 : Using the Pattern

We can then use this pattern in EPL query to identify occurrences of an open door. The query *select*from openDoorPattern* instructs Esper to select all events from the stream that match *openDoorPattern*

Results

Whenever a *openDoor* arrives in the data stream, Esper will evaluate it against the pattern. If the event’s type is “open”, a new event will be generated containing the original event’s information. This new essentially signals the detection of an open door.

Additional Details

- There are several types of EPL patterns.
- Patterns can be combined to create more complex EPL rules.
- Patterns can be extended to meet specific needs.