DejaVu: Declarative Pattern Matching over Live and Archived Streams of Events

Nihal Dindar, Barış Güç, Patrick Lau, Aslı Özal, Merve Soner, Nesime Tatbul @ ETH Zurich

Motivation

• The need for finding contiguous patterns on both live and stored data sequences in Complex Event Processing (CEP) applications such as financial data analysis, supply chain management, and system monitoring.

• Current solution proposals: Pattern matching over live data streams (e.g., SASE, Cayuga) OR Pattern matching over sequences of rows in relational tables (e.g., SQL-TS).

Goal

• To design and implement a scalable complex event processing system that can seamlessly perform pattern detection over both live AND historical streams of events, behind a uniform declarative query interface.

SQL-based Query Language*

```
SELECT notify_theft(tstamp, book_tag)
FROM Books MATCH_RECOGNIZE(
  PARTITION BY TagId
  MEASURES B.Timestamp AS tstamp,
           B.TagId AS book_tag
  ONE ROW PER MATCH
  AFTER MATCH SKIP PAST LAST ROW
  INCREMENTAL MATCH
  PATTERN(A B)
  DEFINE A AS (A.ReaderId = 'Shelf')
         B AS (B.ReaderId = 'Exit')
);
```

* Anonymous, "Pattern Matching in Sequences of Rows", http://asktom.oracle.com/tkyte/row-pattern-recogniton-11-public.pdf, March 2007.

Hybrid Queries: Live Streams ⋈ Archived Streams

Example: Day Trader

```
SELECT min_tstamp_l, symbol_l, min_price_l,
       init_price_a, min_price_a, max_price_a
FROM LiveStock MATCH_RECOGNIZE(
   PARTITION BY Symbol
   MEASURES A.Symbol AS symbol_l
            MIN(B.Timestamp) AS min_tstamp_l,
            MIN(B.Price) AS min_price_l
   ONE ROW PER MATCH
   AFTER MATCH SKIP PAST LAST ROW
   INCREMENTAL MATCH
   PATTERN (A B+)
   DEFINE /* A matches any row */
     B AS (B.Price < A.Price AND
           B.Price <= PREV(B.Price))
   ), ArchivedStock MATCH_RECOGNIZE(
   PARTITION BY Symbol
    MEASURES A.Symbol AS symbol_a,
             A.Price AS init_price_a,
             MIN(B.Price) AS min_price_a,
             LAST(D.Price) AS max_price_a
   ONE ROW PER MATCH
   AFTER MATCH SKIP PAST LAST ROW
   MAXIMAL MATCH
   PATTERN (A B+ C* D+)
   DEFINE /* A matches any row */
     B AS (B.Price < A.Price AND
           B.Price <= PREV(B.Price))
     C AS (C.Price >= PREV(C.Price) AND
           C.Price <= A.Price)
     D AS (D.Price > PREV(D.Price) AND
           D.Price > A.Price)
   )
WHERE symbol_l = symbol_a;
```

Implementation

• When a pattern is detected on the Live Stream ending at time t, look for patterns in the Archived Stream which end before time t.

Optimization

• Patterns previously found on the Archived Stream are stored.

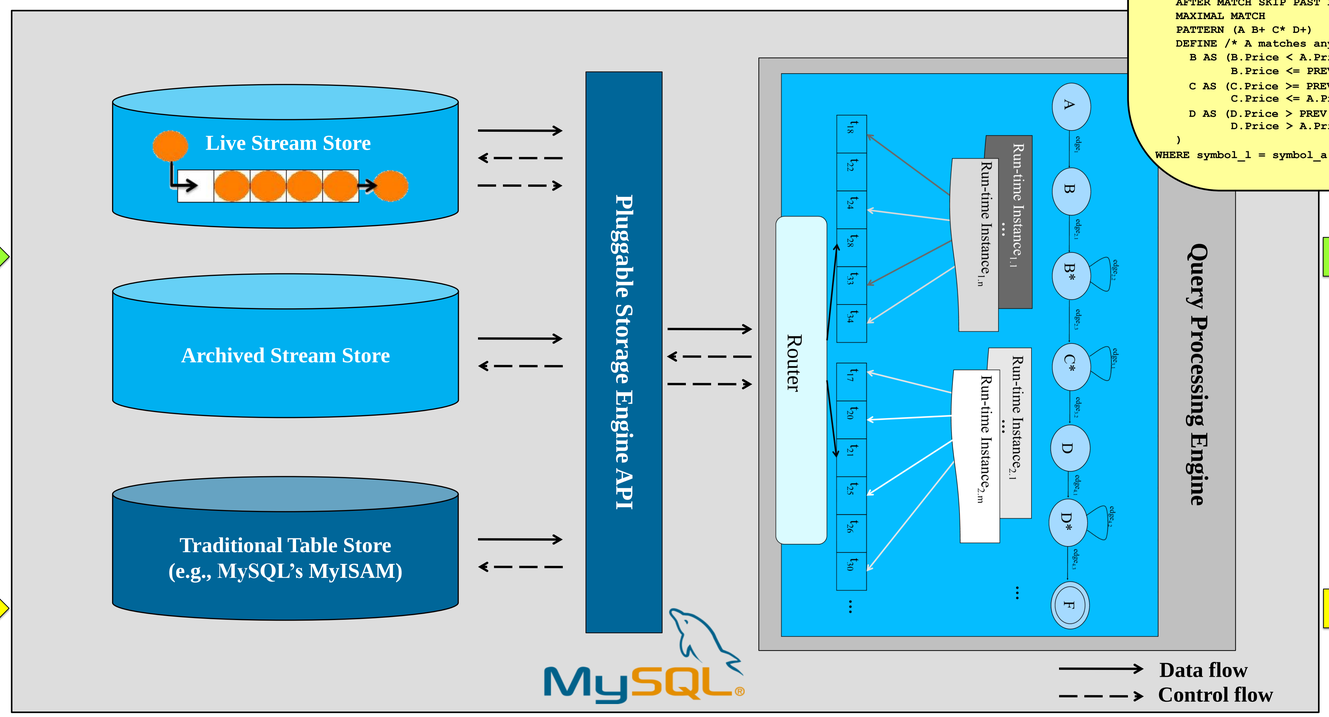
• Next time a new pattern is found on the Live Stream, we only need to process the Archived Stream starting from the last processed position.

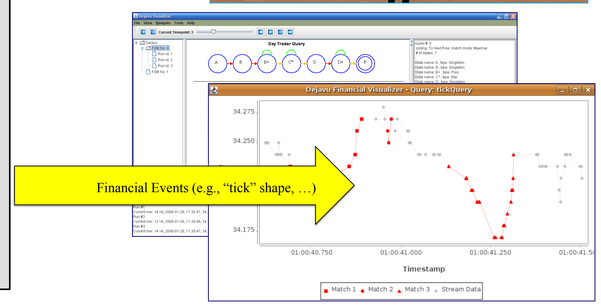
Live Stream

Symbol	Timestamp	Price	Volume
UBS	11:13:23	34.70	800
UBS	11:13:24	34.80	800
...			
UBS	11:13:25	34.82	400
...			
UBS	11:13:27	34.79	800
...			
UBS	11:23:25	34.83	800
UBS	11:24:19	34.84	2000
UBS	11:24:21	34.82	200
UBS	11:24:47	34.78	100
UBS	11:24:49	34.84	200
UBS	11:24:49	34.85	400
UBS	11:24:54	34.80	500
UBS	11:25:24	34.82	700
UBS	11:26:02	34.83	200
UBS	11:26:04	34.92	100
UBS	11:26:05	34.96	100
UBS	11:26:08	34.95	100
UBS	11:27:27	34.92	200
UBS	11:28:03	34.97	500
UBS	11:28:17	34.97	2300

Archived Stream

Symbol	Timestamp	Price	Volume
...			
UBS	11:06:19	34.85	2200
UBS	11:06:21	34.83	200
UBS	11:06:47	34.78	100
UBS	11:06:49	34.81	300
UBS	11:06:49	34.87	400
...			
UBS	11:23:25	34.83	800
UBS	11:24:19	34.84	2000
UBS	11:24:21	34.82	200
UBS	11:24:47	34.78	100
UBS	11:24:49	34.84	200
UBS	11:24:49	34.85	400
UBS	11:24:54	34.80	500
UBS	11:25:24	34.82	700
UBS	11:26:02	34.83	200
UBS	11:26:04	34.92	100
UBS	11:26:05	34.96	100
UBS	11:26:08	34.95	100
UBS	11:27:27	34.92	200
UBS	11:28:03	34.97	500
UBS	11:28:17	34.97	2300





Live Stream Store

• An **in-memory** storage engine that accepts push-based inputs.

• Acts like a **tuple queue**, providing live events into the query processing engine as they arrive from their sources.

• Used to answer **continuous** pattern matching queries.

• Can be accessed either in **"push mode"** or **"pull mode"**.

Archived Stream Store

• A **persistent** storage engine where live events can be fully or selectively materialized for historical access.

• Only allows updates in the form of **appends** and preserves the data order.

• Used to answer **one-time and hybrid** pattern matching queries.

• Can also support the live stream store in dealing with bursts and failures.

Push or Pull?

• The Query Processing Engine can access the live streams in two alternative modes:

- **Push:** By default, each new input event is pushed directly into the corresponding Input Holders via the Router, in order to feed the Query Processing Engine.
- **Pull:** The Query Processing Engine asks for new input events whenever it is ready to process them via the Router.

• DejaVu adaptively switches from Push to Pull when:

$$\frac{\text{QPE input consumption rate}}{\text{Store input push rate}} < \tau$$

DejaVu Query Processing Engine

• DejaVu has been built on the **MySQL** relational database engine; as such, it extends MySQL with a number of key capabilities including:

- the ability to process **continuous queries over streaming data**,
- the ability to process **pattern matching queries**.

• DejaVu represents each pattern with a **Finite State Machine** (FSM), which runs as an integral part of the MySQL query plan.

• Each FSM instance can be at multiple active states at a given time, due to the inherent non-determinism and/or overlapping semantic windows. In this case, multiple FSM instances can share input tuples through **Input Holders**.

• A **Router** component forwards the relevant tuples to each of the Input Holders in an efficient manner.

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich
FNSNF Swiss National Science Foundation
MICS