Event & Processing Time in Table API and SQL

Problem Definition

Many operations of relational queries on streaming data are defined on time:

- Windowed aggregates (GROUP BY or OVER ORDER BY)
- Windowed stream-stream joins
- Joins between streams and “history” tables
- Time-based predicates

Flink’s Table API and SQL support two time modes, event-time and processing-time. In event-time, the associated time of a row depends on an attribute of the row. In processing-time it is the wallclock time when the row arrives at an operator.

In order to define a time-dependent query, we need to refer to the time attribute for event-time or indicate processing time. In either case the time expression must relate to a table. This is important for join queries as the following query shows:

```
SELECT a.amount, r.rate
FROM
  amounts AS a,
  rates AS r
WHERE
  a.currency = r.currency AND
  r.proctime = (
    SELECT MAX(r2.proctime)
    FROM rates AS r2
    AND r2.proctime <= a.proctime)
```

The query joins two streaming tables. Table 1 is a streaming table with amounts in different currencies. Table 2 is a (slowly changing) streaming table of currency exchange rates.
We want to join the amounts stream with the exchange rate of the corresponding currency that is valid when the amounts row arrives, i.e, we have a join condition on the proctime attributes of both tables.

In this document we propose how to specify event and processing-time in Table API and SQL queries.

Processing Time

For processing-time, we define a virtual system attribute. When specifying a query, users refer to the attribute to define processing time semantics. However, the attribute is only used during parsing and validation of Table API and SQL queries and converted into a special expression before optimization. During optimization and translation, we know how to handle the rowtime.

This design has the following benefits.

- Records do not carry any additional data.
- Defining processing time feels very natural because it is just referring to an attribute.

Event Time

The event-time case is harder to handle. In contrast to processing-time, the result of a query specified with event-time semantics does only depend on the input data and the query and must be equivalent to the result of running the same query on the same input data in batch (or streaming) mode.

In Flink's DataStream API, several aspects of event-time are implicitly handled by Flink's runtime:

- Record timestamps are manually assigned once by a timestamp extractor and thenceforth automatically managed
- Record timestamps are hidden metadata and not accessible
- Timestamps for records which result from windowed aggregates are automatically assigned

For the Table API and SQL, the whole semantics of a query must be explicitly spelled out and may not depend on system behavior. This is important in order to comply with the semantics of batch SQL queries.

Let's look at an example again and extend the previous join query. Now we want to first aggregate all amounts of the same currency per day before joining it with the rates stream. With standard SQL this looks like:

```
SELECT
  a.amountSum, r.rate, a.day
FROM
  (
    SELECT
      currency,
      SUM(amount) AS amountSum,
```

```
      CEIL(rowtime TO DAY) AS day
    FROM amounts
    GROUP BY currency, CEIL(rowtime TO DAY)
  ) AS a
  rates AS r
WHERE
  a.currency = r.currency AND
  r.rowtime = (
    SELECT MAX(r2.rowtime)
    FROM rates AS r2
    AND r2.rowtime <= a.day)
```

This query is composed of two parts:

- an inner query which sums the amounts per currency and day (tumbling window)
- an outer query which joins the resulting daily aggregates with the last (closing) rate of the day.

The important part here is that the outer query depends on a time attribute (CEIL(rowtime TO DAY) AS day) which is computed in the inner query. This attribute is the new event-time row of the intermediate table (a). By fully specifying the event-time attribute, the example query can be executed on a stream and on a batch table and will produce identical results (given appropriate watermarks in the stream).

Running the same query on a batch table would not be possible if we would treat event-time as an internal concept or implicitly assign new event-time timestamps to rows as the DataStream API does. Consequently, a users need to be able to explicitly specify the event-time attributes of tables by either forwarding the time attribute of the input table or computing new values if the query combined multiple rows as in GROUP BY aggregates, and joins.

Definition of Event Time Attributes, Timestamps and Watermarks

In order to achieve semantic equivalence for batch and streaming (event-time) queries, users must be able to define attributes to which time dependent operations, such as windows and joins, relate. Flink's event-time operators rely on record timestamps and watermarks. Timestamps are metadata which is attached to each record. Watermarks are special records that are injected into a stream and which tell operators when to trigger computations and clean up state. These watermarks and record timestamps must be properly aligned, i.e., records are expected to arrive (almost) in timestamp order and watermarks follow the same order with a bit of slack to allow for out-of-order timestamped records.

The Table API and SQL require that streams, which are converted into Tables (either from a DataStream or a StreamTableSource) and which should be processed with event-time semantics, have already timestamps and watermarks assigned. A time-dependent query must refer to the existing event-time timestamp. An operation which is performed on the table that generates new records (windows, joins, …) needs to assign event-time timestamps to these

records. As discussed before, the definition of the timestamps must be included in the query in order to have the query semantics completely specified. However, we also must ensure that the relationship between timestamps and watermarks is respected. There are basically two ways to achieve that:

1. Do not change watermarks and only allow timestamps which are aligned with watermarks.
   The goal of this approach is preserve the existing watermarks. Hence, we can only allow new timestamps which are aligned with the existing watermarks. This implies that only a value which is derived from the existing timestamp attribute can be used as a timestamp. Also the logic for how to derive the new timestamp would be restricted, i.e., only certain expressions would be allowed to compute a new timestamp.
   For example, the timestamps of records which originate from a GROUP BY window aggregation must be equal to the end time of the window (see CEIL(rowtime TO DAY) as an example). In fact, Flink's DataStream API assigns records from windowed aggregates the end-time of the window as timestamp.

2. Reassign watermarks according to new timestamps.
   This approach would allow to specify new timestamps and require to reassign new watermarks as well. However, the approach as a few challenges and drawbacks. First, the choice of new timestamps is very limited due to the requirement of increasing timestamps. Timestamps can only be derived from existing attributes. Therefore, only the current timestamp attribute or attributes which have the same order as the current timestamp attribute but are off by a constant are viable options. Eventually, this approach does not add much more freedom compared to the first approach. Moreover, we would need a strategy to assign watermarks such as following the max timestamp by a constant interval (like the BoundedOutOfOrdernessAssigner available in the DataStream API).

Since the second approach does not provide significantly more freedom to users, I would propose to stick with the first approach for now. Later we can still think about relaxing the conditions.

Usage of Event-Time Attributes

Now that we discussed why event-time attributes need to be specified and how they can be specified, we need to discuss how we can represent this in the APIs and during translation and optimization.

Event-time processing is based on actual data. Hence, event-time timestamps should be defined and used just like regular attributes. I see two approaches with advantages and drawbacks of how to integrate them into the API and translation process.

1. Dedicated event-time attribute.
   In this approach, there is a dedicate attribute (configurable per table environment, by default called “rowtime”) which serves as event-time column. Hence, all time-based operations always have to reference this attribute and the event-time timestamp is modified by specifying an attribute with the corresponding name. This approach has the following benefits:
   - We know when an event-time attribute is defined because of its name and can immediately validate that it has a valid value.
   - We know exactly, which attribute is a valid event-time attribute to be used in a window or join. Since we validate that the time attribute is valid when it is defined, we are sure that it is valid.

   Drawbacks are:
   - The schema of tables is constrained, i.e., attribute names cannot be freely chosen. This makes portability of queries to batch tables more difficult.
   - Since the time attribute needs to be specified on a table environment level, all tables in a table environment are affected and queries have a dependency on the table environment configuration.

2. Arbitrary event-time attributes.
   Another approach would be to allow any name for event-time attributes. The attribute would be initially defined when defining the schema of the table, i.e., either when converting a DataStream into a Table or by a StreamTableSource.
   The benefit of this approach is that users would have full flexibility in choice of attribute names. Queries become more portable to batch tables.
   The drawback is the more complex implementation. Whenever an attribute is used as a time attribute, we have to check that it is a valid time attribute, i.e., we have to backtrack its origin or annotate all valid time attributes with a certain flag. We would need to check how we can implement this in

In my opinion, we should not restrict the choice of attribute names even though it is harder to implement.

Summary

- For processing time, we add a virtual attribute which is translated into an expression after validation and before optimization
- For event time
  - We require an initial definition of a time attribute.
  - Time attributes are regular attributes but requires some validity checks.
  - The time attribute can be transformed into another valid time attribute (depending on the operation). Time attribute transformations must be aligned with existing watermarks and are (initially) restricted.

- When defining a time-dependent operation, we check that the attribute is a valid time attribute.