A Relational Algebra for Streaming Tables Living in a Temporal Database World

Fabio Grandi[1], Federica Mandreoli[2], Riccardo Martoglia[3], and Wilma Penzo[4]

1 **DISI - University of Bologna, Bologna, Italy**
`fabio.grandi@unibo.it`
2 **FIM - University of Modena e Reggio Emilia, Modena, Italy**
`federica.mandreoli@unimo.it`
3 **FIM - University of Modena e Reggio Emilia, Modena, Italy**
`riccardo.martoglia@unimo.it`
4 **DISI - University of Bologna, Bologna, Italy**
`wilma.penzo@unibo.it`

Abstract

The recently introduced streaming table concept, a fully native representation of streaming data inside a DBMS, enabled modern data-intensive applications with one-time queries (OTQs) and continuous queries (CQs) capabilities on both streaming and standard relational tables. In this paper, we fully acknowledge the temporal nature of streaming tables and we propose to go one step further and integrate them in a temporal DBMS context, where time management is native. Our aim is to break the traditional barrier between the streaming and the temporal worlds, offering complete interoperability between streams and temporal data. To this end, we present a continuous temporal algebra supporting both OTQs and CQs seamlessly on streaming, standard and temporal relational tables. We further show how the transition from continuous to one-time semantics can be managed by defining suitable translation rules, which can also be used as a basis for the implementation of the proposed continuous algebra in a temporal DBMS.

1998 ACM Subject Classification H.2.1 Data Models, H.2.3 Query Languages

Keywords and phrases Continuous queries, Data streams, Relational algebra, Temporal DB

Digital Object Identifier 10.4230/LIPIcs.TIME.2017.15

1 Introduction

Modern data-intensive applications, including for instance advanced surveillance (e.g., financial market enforcement), monitoring applications (e.g., air quality monitoring, Intelligent Transportation Systems - ITSs), military applications (e.g., platoon tracking), network applications (e.g., intrusion detection), more and more often require an increasingly wider range of data management capabilities in order to be fully supported. First of all, they need to manage very large quantities of continuously streaming data over which complex continuous queries (CQs) have to be efficiently executed. Unlike typical applications relying on Data Stream Management Systems (DSMSs) [23], such as sensor-data analysis and geospatial services, not only needs the most recent data to be retained and managed, but the whole incoming streams have to be stored as historical data in order to make them fully persistent and available to future analysis. Moreover, there is the need to be able to easily perform everything we are accustomed to do in a traditional Data Base Management Systems (DBMSs) context, including one-time queries (OTQs) also involving standard relational data, and possibly in a

temporal DBMS context, allowing for complex temporal analytics and management of data versioning.

Being able to query recent, historical and non-temporal relational data by means of both CQs and OTQs in a uniform and powerful way, and with the expressive power of standard RDBMS's languages and algebras, is certainly a very strong requirement. The very diversified and lively panorama of data management systems, including streaming extensions to traditional DBMSs (e.g., [10, 11, 21]), big data processing engines, big data stores and stream processing frameworks (e.g., [8, 22]), is still trying to fully address it. One first step towards this objective has been performed in [7], where we proposed a fully native representation of streaming data into a DBMS. The introduction of a new kind of table, called the *streaming table*, allowed us to offer a persistent structure managing both historical and streaming data in a way completely transparent to users. Streaming tables can also be straightforwardly involved in OTQs and CQs, possibly together with standard relational tables, thus successfully merging the two worlds of RDBMSs and streaming data management.

Heading further in this direction, in this paper we aim at breaking another barrier: the one currently present between streaming and temporal data management. Even if streaming data is inherently temporal in nature, current systems and research proposals seem unable to build on this: on the one hand, temporal RDBMSs are not able to deal with streams, on the other hand current DSMSs and stream processing frameworks only provide limited temporal querying possibilities or do not deal with versioned data at all [8, 22]. By fully acknowledging the temporal nature of streaming data, we propose to integrate streaming tables in a temporal context where time management is native. Our final aim is to offer temporal RDBMSs' querying capabilities and data versioning even on streaming data, without overturning the common understanding of streaming data and CQs. In this way, data can be easily managed and queried by benefitting from the best of the relational, streaming and temporal worlds.

In this paper, we lay the semantic foundation for our objective by introducing a continuous temporal algebra for streaming, temporal, and standard tables. The proposed algebra, denoted as $\mathcal{CTA}$, extends with windowing operators a temporal algebra $\mathcal{TA}$ with well-defined semantics. Window operators take streaming tables as input and produce data snapshots at subsequent validity time points. Continuous and one-time queries are specified as algebraic expressions over the three different kinds of tables. In particular, OTQs can be formulated as $\mathcal{TA}$ expressions and their semantics relies on the traditional $\mathcal{TA}$ definition. CQs can be formulated as $\mathcal{CTA}$ expressions over standard and temporal tables and windowing expressions on streaming tables, and their semantics relies on a sampling operator that evaluates continuous temporal expressions at the required time points. In this way, we introduce a sort of *on-demand semantics*, where the CQs are executed when query results are needed and required data are available. This approach is highly flexible in that it allows to combine in a unified framework several continuous query features proposed in the literature (e.g., real time and historical analytics, backward and forward sliding windows, tumbling and hopping windows) but for which a well-founded semantics and full implementation agenda is still lacking, and to interoperate streaming data with non-temporal data and archival data stored in temporal tables in a consistent way.

As an application example in the context of financial market surveillance, we consider tracking of *insider trading* activities, that is the buying or selling of a security (e.g., stock options) while in possession of nonpublic material information (e.g., up/downgrade by a rating agency, unexpected revisions to earning results or projections, mergers and acquisitions news) about the security [1]. Since insider trading undermines investor confidence in the fairness

and integrity of the financial markets, control bodies like the SEC have the detection and prosecution of insider trading violations as one of their market surveillance and enforcement priorities. Whereas early (i.e., when the material information is not yet publicly available) detection of insider trading patterns could allow to prevent frauds [14], their *a posteriori* (i.e., when the material information is of public domain) verification is most important for triggering and pursuing prosecution of illegal activities. Assume that option trading data are available through the streaming table `OPTION_TRADES`, with schema

`OPTION_TRADES(OPTION,STOCK,CLASS,STRIKE,EXPIR,CONTRACTS|`T`)`

automatically fed by the stock market information system, containing data concerning the negotiated option, the underlying stock, the trade timestamp (i.e., the implicit attribute T defined with a granularity of one second), the option class (call or put), strike price and expiration, and the number of contracts traded. Further, assume that relevant news are selected by several sources (e.g., press releases and financial news stories) and stored in a temporal table `NEWS`, with schema

`NEWS(STOCK,TYPE,SOURCE|`T`)`

containing information about the publication date (i.e., the implicit attribute T defined with a granularity of one day), the mentioned stock, the news type and source. In order to trigger an investigation procedure, an anomalous trading volume of an option, preceding by at most one week the public release of some relevant news concerning the underlying stock, needs to be identified. As anomalous volume, we consider a daily trading volume ten times higher than the average daily volume over the past month. Following our proposal, the suspect stock-day pairs deserving further investigations could be easily retrieved via a continuous query running at the beginning of each trading day and performing the temporal join between the relevant time windows of the streaming table `OPTION_TRADES` and the standard table `NEWS`. Notice that such a detection query could not be executed in a "classical" stream management system, because, when the windows used for computing volumes are evaluated, the relevant news have not been published yet. Hence, stream data must be stored in a streaming table for our purpose, and time windows evaluated in a delayed mode (i.e., when all relevant news will be available) as actually supported by the CQ on-demand semantics. The temporal join involved in the query will produce temporally consistent solutions over the past time window outputs of the streaming table `OPTION_TRADES` and the temporal table `NEWS`.

After revising the notion of streaming table (Sec. 2), this paper provides the following contributions:

- it introduces the continuous temporal algebra $\mathcal{CTA}$ supporting both OTQs and CQs seamlessly on streaming, standard and temporal relational tables (Sec. 3 and Sec. 4);
- it proposes a correct translation of the continuous temporal model presented so far into the temporal model where continuous queries are transformed into standard temporal queries, making it possible to instantiate the framework in the context of a standard temporal RDBMS (Sec. 6).

The paper is complemented by Sec. 5, expressing the reference example in the $\mathcal{CTA}$ algebra and Sec. 7, comparing our contribution with the state of the art and drawing conclusions.

2 Preliminaries

The continuous temporal data model we propose relies on a multi-temporal relational model [17], where temporal and non-temporal standard tables coexist, extended with streaming tables. In this Section we provide some preliminaries for its specification by reviewing the notion of streaming table, first introduced in [7].

First of all, we assume a discrete, ordered and unbounded time domain $\mathcal{T} = \{0, 1, 2, \ldots, \infty\}$ composed of *chronons* [18], where 0 stands for the earliest time. A chronon is a non-decomposable time interval of fixed unit duration used to represent time instants in the discrete model. We further assume that $\mathcal{T}$ has the semantics of *valid time* [18]. In order to represent a duration of time, we assume *time spans* [18] belong to a domain $\mathcal{I}$ composed of all possible multiples of a chronon duration (including the unbounded value ∞).

As far as the temporal model is concerned, a temporal relation R with explicit schema $R(A_1, \ldots, A_n)$, with $A_i \in \mathcal{A}$ $(1 \leq i \leq n)$ where $\mathcal{A}$ is the set of attribute names, is represented as $R(A_1, \ldots, A_n|T)$ where T is the implicit timestamp attribute with domain $\mathcal{T}$. If r is a tuple from R then $r.A_i$ denotes the value of A_i in r and $T(r)$ denotes the tuple timestamp. Moreover, given any time instant $t \in \mathcal{T}$, we denote with R^t the content of R at time t. Notice that we assume here an *abstract temporal database*, according to the terminology introduced in [12], to be used as a representation-independent data model. As far as non-temporal tables are concerned, we assume they are virtually converted to temporal tables to be interoperated with temporal tables and streaming tables by using a suitable temporal conversion map [9]. In particular, we assume each non-temporal table R to be virtually converted to a temporal table $R' = \{(r, \tau) \,|\, r \in R, \tau \in \mathcal{T}\}$.

As far as streams are concerned, we adopt the definition of continuous data stream (or simply stream) provided in [4], that is a potentially infinite stream of timestamped relational tuples having a fixed schema. A *streaming table* [7] is a relational table where streaming data enter and turn historical by remaining stored for a user-defined long period, ideally forever. Any streaming table inherits the temporal nature of the data it stores. Specifically, it is an event table [26, Ch. 16], that is a special kind of temporal table that stores event data and their occurrence time, for a limited time span named *historical period*. As time goes by, we assume the oldest data exiting the historical period are subject to *vacuuming* [26, Ch. 23].

▶ **Definition 1** (Streaming table)**.** A streaming table S with explicit schema $S(A_1, \ldots, A_n)$ and historical period $hp \in \mathcal{I}$ is an event table, denoted as S_{hp}, with schema $S(A_1, \ldots, A_n|T)$, where T is the implicit timestamp attribute. The content of S_{hp} at the time instant $t \leq now$ (where now represents current time), i.e. S^t_{hp}, is the set of tuples such that the timestamp of each tuple s satisfies $T(s) \geq \max(t - hp, 0)$. If positive, $t - hp$ represents the chronon preceding t by a time span of hp chronons. A streaming table is subject to *continuous writes* in timestamp order, that is for any s_1 and s_2 in S^t_{hp}, $T(s_1) < T(s_2)$ iff s_1 arrived before s_2.

For ease of notation, in the following, whenever possible, we will use S in place of S_{hp}. In practice, in order to implement this insertion semantics, systems cope with out-of-order and skewed inputs. Interested readers can refer to [27] for an in-depth discussion of this aspect. In this paper we assume an input manager that guarantees in-order tuple arrival.

In the following, let $\mathcal{R}$ be the set of all temporal tables and $\mathcal{S}$ be the set of all streaming tables. Notice that $\mathcal{S} \subseteq \mathcal{R}$, as streaming tables are a special kind of temporal tables (one effect of the insertion semantics is that timestamps in a streaming table are always bounded by the current time now, whereas temporal tables may also contain timestamps greater than now to represent proactively inserted future data).

3 Querying streaming tables with OTQs: the temporal algebra $\mathcal{TA}$

In OTQs a streaming table is dealt with as a standard (event) table and it undergoes queries expressed in a relational algebra $\mathcal{TA}$ with the following temporal operators: selection σ^T, projection π^T, Cartesian product $\times^T$, union $\cup^T$, difference $-^T$, and grouping ϑ^T. Each of

these temporal operators is a generalization of a standard relational operator where the T-superscript does not appear (derived operators, like the join $\bowtie^T$, can also be considered as usual). The definition of these operators is borrowed from [13] and, thus, $\mathcal{TA}$ supports a sequenced semantics in terms of extended snapshot reducibility. To this purpose, in order to support operators with predicates and functions that reference the timestamp, the additional extend operator ε_U copies the timestamp T to the additional attribute U. Notice that τ_t is an "extended" timeslice operator that maintains the timestamps, such that its result is still a temporal relation representing the *snapshot* valid at time t. Non temporal operators may be needed to express non-sequenced parts of queries (e.g., to join data belonging to different snapshots).

The semantics of the temporal algebra operators is shown in Table 1, where if (r, τ) is a tuple of a temporal relation with explicit schema $R(X)$ we consider r (with schema X) its explicit part and τ the tuple timestamp, $\mathcal{P}$ is the set of all well-defined predicates p over the explicit attributes X of R, $B \subset X$ is a subset of schema attributes, $\mathcal{F} = \{f_1, \ldots, f_k\}$ is a set of aggregation functions, and $\circ$ is a tuple concatenation operator. Notice that the Cartesian product applied to at least one streaming table returns a streaming table, the union operator returns a streaming table when both operands are streaming tables and the difference operator returns a streaming table when the first operand is a streaming table.

Operator	Signature	Operator semantics
Selection	$\sigma^T : \mathcal{R} \times \mathcal{P} \to \mathcal{R}$	$\sigma^T_p(R) := \{(r,\tau) \mid (r,\tau) \in R \wedge p(r)\}$
Projection	$\pi^T : \mathcal{R} \times 2^{\mathcal{A}} \to \mathcal{R}$	$\pi^T_B(R) := \{(r.B,\tau) \mid (r,\tau) \in R\}$
Cart. prod.	$\times^T : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$	$R_1 \times^T R_2 := \{(r_1 \circ r_2,\tau) \mid (r_1,\tau) \in R_1 \wedge (r_2,\tau) \in R_2\}$
Union	$\cup^T : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$	$R_1 \cup^T R_2 := \{(r,\tau) \mid (r,\tau) \in R_1 \vee (r,\tau) \in R_2\}$
Difference	$-^T : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$	$R_1 -^T R_2 := \{(r,\tau) \mid (r,\tau) \in R_1 \wedge (r,\tau) \notin R_2\}$
Grouping	$\vartheta^T : \mathcal{R} \times 2^{\mathcal{A}} \times 2^{\mathcal{F}} \to \mathcal{R}$	${}_B\vartheta^T_F(R) := \{(r.B \circ Z,\tau) \mid (r,\tau) \in R \wedge$ $r_g = \{(r',\tau) \in R \mid r'.B = r.B\} \wedge Z = (f_1(r_g), \ldots, f_h(r_g))\}$
Extend	$\varepsilon : \mathcal{R} \times \mathcal{A} \to \mathcal{R}$	$\varepsilon_U(R) := \{(r \circ U,\tau) \mid (r,\tau) \in R \wedge U = \tau\}$
Timeslice	$\tau : \mathcal{R} \times \mathcal{T} \to \mathcal{R}$	$\tau_t(R) := \{(r,\tau) \mid (r,\tau) \in R \wedge \tau = t\}$

Table 1 $\mathcal{TA}$ operator semantics

We assume readers are familiar with the syntax and semantics of relational queries[1] and we only provide a concise yet informal semantics of OTQs.

▶ **Definition 2** (Semantics of one-time queries over streaming and temporal tables)**.** Given a $\mathcal{TA}$ expression $Q = E_{\mathcal{TA}}\ (S_1, \ldots, S_n, R_1, \ldots, R_m)$, over n streaming tables $S_1, \ldots, S_n$, with $n \geq 1$, and m temporal tables $R_1, \ldots, R_m$, with $m \geq 0$, its semantics is the result of the semantics of the involved temporal operators for which snapshot reducibility holds, that is for each $t \in \mathcal{T}$: $\tau_t(Q) = E_{\mathcal{TA}}\ (\tau_t(S_1), \ldots, \tau_t(S_n), \tau_t(R_1), \ldots, \tau_t(R_m))$.

4 Querying streaming tables with CQs: the continuous temporal algebra $\mathcal{CTA}$

As we have mentioned, streaming tables can also be involved in CQs. A continuous query Q^c is a query that is issued once, and then logically runs continuously until terminated by

[1] Interested readers can refer to [3] for an in-depth study.

the user. Any streaming table S referenced in a continuous query must be accessed through a sliding window w that specifies the boundaries of the range of tuples in S to be used for query evaluation. The continuous temporal algebra $\mathcal{CTA}$ we propose extends the set of windowing operators usually adopted in the streaming context [24], by generalizing their semantics to generate and operate on, possibly through aggregation, *sequences* of windows instead of single windows. Thus, $\mathcal{CTA}$ extends the temporal algebra $\mathcal{TA}$ ($\mathcal{TA} \subseteq \mathcal{CTA}$) with the introduction of such generalized windowing operators.

For ease of notation, we start introducing some utility operators. The definitions of $\mathcal{CTA}$ operators follow.

4.1 Utility operators

The following operators provide useful transformations of streaming tables.

▶ **Definition 3** (Substreaming)**.** The substreaming operator $\text{Sub} : \mathcal{S} \times \mathcal{T}^2 \to \mathcal{S}$ restricts a streaming table $S \in \mathcal{S}$ to only tuples such that their timestamp belongs to an interval $[t_1, t_2]$:

$$\text{Sub}_{[t_1,t_2]}(S) := \{(s,\tau) \,|\, (s,\tau) \in S \wedge t_1 \leq \tau \leq t_2\}.$$

For instance, the expression $\text{Sub}_{[\text{2016-01-01 00:00:00, 2016-12-31 23:59:59}]}(\texttt{OPTION\_TRADES})$ builds from `OPTION_TRADES` a streaming table containing the trades executed in 2016 only.

▶ **Definition 4** (Streaming Table Partition)**.** The partitioning operator $\zeta : \mathcal{S} \times 2^{\mathcal{A}} \to 2^{\mathcal{S}}$ partitions the streaming table $S \in \mathcal{S}$ into a set of streaming tables containing the tuples of $\mathcal{S}$ grouped by their attributes in B (as for the SQL group by mechanism, a partition is created for each combination of the values of the attributes $B_1, \ldots, B_k$ in B):

$$\zeta_B(S) := \{S' \,|\, (s,\tau) \in S \wedge S' = \{(s',\tau') \,|\, (s',\tau') \in S \wedge s'.B = s.B\}\,\}.$$

For instance, the expression $\zeta_{\texttt{CLASS}}(\texttt{OPTION\_TRADES})$, partitioning `OPTION_TRADES` with respect to the values of `CLASS`, evaluates to a set containing two streaming tables, one containing all the trades on call-type options and the other containing all the trades on put-type options present in `OPTION_TRADES`.

4.2 $\mathcal{CTA}$ operators

In order to support continuous queries over streaming tables, $\mathcal{CTA}$ introduces two classes of operators: the former includes sliding window operators, the latter includes window flattening and aggregation operators. For the sake of clarity, we give an intuition of these two classes of operators that work in a complementary fashion: sliding window operators generate a sequence of portions of a given streaming table S according to a sliding window specification, thus resulting into a sequence of windows over S; window flattening operators and window aggregation operators reduce the result of a sliding window operator to a streaming table. Both classes of operators include one *standard* and one *partitioned* version of each operator.

4.3 Sliding window operators

In accordance with commonly adopted definitions of sliding windows [24], sliding window operators in $\mathcal{CTA}$ are time-based and count-based. Standard sliding window operators apply to a streaming table S and generate a temporal relation of streaming tables, each made of portions of S. More precisely, such a temporal relation is a streaming table, thus resulting in a streaming table of streaming tables (we denote by $\mathcal{S}^*$ the set of all possible streaming tables

of streaming tables). Definitions of (backward and/or forward) standard sliding window operators are provided below.

▶ **Definition 5** (Time-based Sliding Window)**.** The time-based sliding window operator $w^{\text{time}} : \mathcal{S} \times \mathcal{I}^2 \to \mathcal{S}^*$ over a streaming table $S \in \mathcal{S}$ creates a streaming table of streaming tables with a window size of duration $\omega_1 \geq 0$ before and $\omega_2 \geq 0$ after the timestamps around which it is computed:

$$w^{\text{time}}_{[\omega_1,\omega_2]}(S) := \{(S', \tau) \,|\, S' = \text{Sub}_{[\tau-\omega_1,\tau+\omega_2]}(S)\}.$$

For instance, the expression $w^{\text{time}}_{[1\text{week},0]}(\texttt{OPTION\_TRADES})$ defines a streaming table whose tuples, for each timestamp τ, are in turn streaming tables extracted from `OPTION_TRADES` by restricting it to the 1-week wide time window preceding τ.

▶ **Definition 6** (Count-based Sliding Window)**.** The count-based sliding window operator $w^{\text{count}} : \mathcal{S} \times \mathbb{N}^2 \to \mathcal{S}^*$ over a streaming table $S \in \mathcal{S}$ creates a streaming table of streaming tables with a window containing the closest $n_1 \geq 0$ tuples valid before and the closest $n_2 \geq 0$ tuples valid after the timestamps around which it is computed:

$$w^{\text{count}}_{[n_1,n_2]}(S) := \{(S', \tau) \,|\, S' = \text{Sub}_{[t_1,t_2]}(S), |\text{Sub}_{[t_1,\tau-1]}(S')| = n_1, |\text{Sub}_{[\tau+1,t_2]}(S')| = n_2\}.$$

In the definition of (S', τ) above, in case $|\text{Sub}_{[t_1+1,\tau-1]}(S')| < n_1$ but $|\text{Sub}_{[t_1,\tau-1]}(S')| = n'_1 > n_1$, we assume $n'_1 - n_1$ tuples all valid at t_1 are, as customary in the streaming context, non-deterministically chosen and removed from S'. Similarly, in case $|\text{Sub}_{[\tau+1,t_2-1]}(S')| < n_2$ but $|\text{Sub}_{[\tau+1,t_2]}(S')| = n'_2 > n_2$, we assume $n'_2 - n_2$ tuples all valid at t_2 are non-deterministically chosen and removed from S'. For instance, the expression $w^{\text{count}}_{[1,1]}(\texttt{OPTION\_TRADES})$ defines a streaming table whose tuples, for each timestamp τ, are in turn streaming tables extracted from `OPTION_TRADES` and containing one tuple immediately preceding τ, the tuples valid at τ (if they exist), and one tuple immediately following τ.

The partitioned version of each sliding window operator applies to a streaming table S and generates a set of streaming tables of streaming tables, resulting from the application of the corresponding standard sliding window operator to each streaming table obtained by the partition of S according to a given set of attributes B. The operators' definitions follow.

▶ **Definition 7** (Time-based Partitioned Window)**.** The time-based partitioned sliding window operator $W^{\text{time}} : \mathcal{S} \times 2^{\mathcal{A}} \times \mathcal{I}^2 \to 2^{\mathcal{S}^*}$ over a streaming table $S \in \mathcal{S}$ creates a set (of streaming tables of streaming tables) composed of the time-based sliding windows (with a window size of duration $\omega_1 \geq 0$ before and $\omega_2 \geq 0$ after the timestamps around which it is computed) computed over the streaming tables into which S is partitioned according to the attributes in B:

$$W^{\text{time},B}_{[\omega_1,\omega_2]}(S) := \{w^{\text{time}}_{[\omega_1,\omega_2]}(S') \,|\, S' \in \zeta_B(S)\}.$$

For instance, the expression $W^{\text{time},\texttt{OPTION}}_{[0,1\text{hour}]}(\texttt{OPTION\_TRADES})$, involving a time-based partitioned window, defines a set of streaming tables, each one corresponding to a different option, composed of the streaming tables whose tuples timestamped with τ are the streaming tables containing the trades involving that option negotiated in the hour that follows τ.

▶ **Definition 8** (Count-based Partitioned Window)**.** The count-based partitioned sliding window operator $W^{\text{count}} : \mathcal{S} \times 2^{\mathcal{A}} \times \mathbb{N}^2 \to 2^{\mathcal{S}^*}$ over a streaming table $S \in \mathcal{S}$ creates a set (of streaming tables of streaming tables) composed of the count-based sliding window (with a window containing the closest $n_1 \geq 0$ tuples valid before and the closest $n_2 \geq 0$ tuples valid after the timestamps around which it is computed) computed over the streaming tables into which S is partitioned according to the attributes in B:

$$W^{\text{count},B}_{[n_1,n_2]}(S) := \{w^{\text{count}}_{[n_1,n_2]}(S') \,|\, S' \in \zeta_B(S)\}.$$

For instance, the expression $W^{\text{count},\texttt{STOCK}}_{[10,0]}(\texttt{OPTION\_TRADES})$, involving a count-based partitioned window definition, denotes a set of streaming tables, each for a different stock, composed of the streaming tables whose tuples timestamped with τ are the streaming tables containing the 10 most recent option trades preceding τ concerning that stock.

4.4 Window flattening operators

Window flattening operators allow for *normalizing* a streaming table (or a set of streaming tables) of streaming tables resulting from a time-based or a count-based sliding window operator to a flat streaming table. As for sliding window operators, the window flattening operator is introduced both in its standard and in its partitioned version.

▶ **Definition 9** (Window Flattening)**.** The window flattening operator $\varphi : \mathcal{S}^* \to \mathcal{S}$ over a streaming table of streaming tables w creates a streaming table composed of the tuples belonging to the streaming tables in w valid at the time at which the flattening is computed:

$$\varphi(w) := \{(\varepsilon_U(s), \tau) \,|\, (S, \tau) \in w \wedge s \in S\}.$$

For instance, the expression $\varphi(w^{\text{time}}_{[1\text{hour},0]}(\texttt{OPTION\_TRADES}))$, involving a window flattening operator, builds a streaming table whose tuples with timestamp τ are all the tuples belonging to the streaming table $w^{\text{time}}_{[1\text{hour},0]}(\texttt{OPTION\_TRADES})$ valid at τ, that is belonging to the 1-hour wide time window of `OPTION_TRADES` preceding τ. Such tuples come out all timestamped with τ in the result but preserve the value of the original timestamp they had in `OPTION_TRADES` converted into an explicit attribute U.

▶ **Definition 10** (Partitioned Window Flattening)**.** The partitioned window flattening operator $\Phi : 2^{\mathcal{S}^*} \to \mathcal{S}$ over a set of streaming tables of streaming tables W creates a streaming table composed of the tuples belonging to the streaming tables in $w \in W$ valid at the time at which the flattening is computed:

$$\Phi(w) := \{(\varepsilon_U(s), \tau) \,|\, w \in W \wedge (S, \tau) \in w \wedge s \in S\}.$$

For instance, the expression $\Phi(W^{\text{count},\texttt{OPTION,CLASS}}_{[4,0]}(\texttt{OPTION\_TRADES}))$, involving a partitioned flattening operator, retrieves the data necessary to display, for each time point and for each stock option, a book with the five latest put trades and the five latest call trades.

4.5 Window aggregation operators

Window aggregation operators are defined to compute aggregate data over time-based or count-based sliding windows, according to a set of aggregation functions $\mathcal{F}$. As for operators above, both standard and partitioned versions of the window aggregation operator are provided.

▶ **Definition 11** (Window Aggregation)**.** The sliding window aggregation operator $\vartheta : \mathcal{S}^* \times 2^{\mathcal{F}} \to \mathcal{S}$ over a streaming table of streaming tables w creates a streaming table having as attributes the values of the aggregates in $F = \{f_1, \ldots, f_h\}$ calculated over the streaming table in w valid at the time at which the aggregation is computed:

$$\vartheta_F(w) := \{(Z, \tau) \,|\, (S, \tau) \in w \wedge Z = (f_1(S), \ldots, f_h(S))\}.$$

The window aggregation operator ϑ can be used in queries for computing aggregate data over time-based or count-based sliding windows. For each time point τ, aggregates can be computed over the timestamped tuples belonging to the streaming table S in w valid

at time τ; aggregate functions `MIN`, `MAX`, `COUNT` (and `SUM`, `AVG` if numeric), `FIRST-VALUE`, `LAST-VALUE`, `NTH-VALUE(n)`, can be used on the explicit attributes of S, whereas aggregate functions `FIRST`, `LAST`, `DURATION` can be used on the timestamps of S.

For instance, the expression $\vartheta_{\texttt{DURATION}}(w^{\text{count}}_{[9,0]}(\texttt{OPTION\_TRADES}))$ returns, for each time point, the width of the time window containing the 10 most recent option trades.

▶ **Definition 12** (Partitioned Window Aggregation)**.** The partitioned sliding window aggregation operator $\Theta : 2^{\mathcal{S}^*} \times 2^{\mathcal{A}} \times 2^{\mathcal{F}} \to \mathcal{S}$ over a set of streaming tables of streaming tables W creates a streaming table having as attributes the grouping attributes in B and the values of the aggregates in $F = \{f_1, \ldots, f_h\}$ calculated over the streaming tables w belonging to W valid at the time at which the aggregation is computed:

$$_B\Theta_F(W) := \{(S.B \circ Z, \tau) \,|\, w \in W \wedge (S, \tau) \in w \wedge Z = (f_1(S), \ldots, f_h(S))\}$$

The partitioned window aggregation operator Θ can be used in queries for computing aggregate data over partitioned time-based or count based sliding windows. Also in this case, aggregate functions acting on explicit attributes or timestamps can be used.

For instance, the expression $\texttt{EXPIR}\Theta_{\texttt{COUNT(OPTION)}}(\texttt{EXPIR}W^{\text{time}}_{[0.5\text{hour},0.5\text{hour}]}(\texttt{OPTION\_TRADES}))$, at each timepoint and for each expiration date, returns the number of options with that expiration date traded in a 1-hour wide time window centered around the timepoint.

Notice that the four types of sliding window operators (w^{time}, w^{count}, W^{time}, W^{count}) can be freely declared in querying streaming tables but they can be used in an algebraic expression only in combination with either a window flattening operator (φ, Φ) or a window aggregation operator (ϑ, Θ), which always produce a streaming table. Formally, a windowing expression applied to a streaming table S is of the form $\alpha(\omega(S))$ where the possible combinations are shown in Table 2. Notice that, standard (resp., partitioned) sliding window operators can only be combined with their stardard (resp., partitioned) window flattening or window aggregation counterparts. These constraints ensure that the value of continuous expressions augmenting $\mathcal{TA}$ is always a streaming table, so that the resulting continuous algebra $\mathcal{CTA}$ is closed with respect to (streaming and) temporal tables.

α	ω	legal $\mathcal{CTA}$ expressions involving windows
$\varphi \mid \vartheta$	$w^{\text{time}} \mid w^{\text{count}}$	$\varphi(w^{\text{time}}_{[\omega_1,\omega_2]}(S))$, $\varphi(w^{\text{count}}_{[n_1,n_2]}(S))$, $\vartheta_F(w^{\text{time}}_{[\omega_1,\omega_2]}(S))$, $\vartheta_F(w^{\text{count}}_{[n_1,n_2]}(S))$
$\Phi \mid \Theta$	$W^{\text{time}} \mid W^{\text{count}}$	$\Phi(W^{\text{time},B}_{[\omega_1,\omega_2]}(S))$, $\Phi(W^{\text{count},B}_{[n_1,n_2]}(S))$, $_B\Theta_F(W^{\text{time},B}_{[\omega_1,\omega_2]}(S))$, $_B\Theta_F(W^{\text{count},B}_{[n_1,n_2]}(S))$

Table 2 $\mathcal{CTA}$ operators combinations

4.6 Supporting continuous queries

In order to support continuous queries, a sampling operator is formally introduced to evaluate an algebraic expression expressed in the continuous temporal algebra $\mathcal{CTA}$ at the required time points. In line with many CQ specification syntaxes (e.g. [4]), we assume a continuous query is always equipped with a *slide* parameter *sl* representing the query evaluation period, and with a further optional *alignment* parameter *a* specifying the position of the evaluation point within the evaluation period. Moreover, we also consider a *delay* parameter δ specifying that the evaluation of the query at time t has actually to be executed at time $t + \delta$. The slide parameter can be either a user-supplied time span or the special parameter `REALTIME`, that means that the query is re-evaluated as new tuples arrive. The alignment value is expressed

as a period of time to be counted from the beginning of the time granules representing the evaluation periods (and is ignored in case sl=`REALTIME`). Parameters sl, a and δ used for sampling $\mathcal{CTA}$ expressions allow to generalize the usage of the so-called *tumbling windows* (and *hopping windows*) for producing continuous query results.

▶ **Definition 13** (Sampling Operator)**.** At execution time t, the sampling operator $\xi : \mathcal{CTA} \times \mathcal{T} \times \mathcal{I}^4 \to \mathcal{S}$, with an historical period parameter hp, a sliding parameter sl, an alignment parameter a, causes the evaluation of the continuous algebra expression $E \in \mathcal{CTA}$ at time points $t_0, t_1, \ldots, t_k \leq t$ only, where $t_i = (\lceil \frac{t-hp-a}{sl} \rceil + i) \cdot sl + a$. If a delay parameter δ is specified, it forces the evaluation of the expression E to be actually executed at time $t + \delta$:

$$\xi^{t,\delta}_{hp,sl,a}(E) := \bigcup_{i=0}^{k:t_k \leq t} \tau_{t_i}(E^{t+\delta})$$

For example, if sl="1 day", the continuous execution must produce one result per day: if the alignment parameter is a="30 minutes", the results are produced each day at "00:30" in the morning, whereas if the alignment parameter is a="16 hours", the results are produced each day at 4 p.m.. Notice that different results are produced with respect to the desired alignment, since time windows are defined with reference to the execution times, which depend on the alignment. For instance, assuming daily trading hours range from 9 a.m. to 4 p.m., the sliding window $w^{\text{time}}_{[1\text{day},0]}$(`OPTION_TRADES`) executed via a sampling with sl="1 day" and a="6 hours" includes all the trades executed the day before (from 9 a.m. to 4 p.m.) to contribute to a result produced daily at 6 a.m., but if it were executed with a="12 hours" it would include all the trades executed in the afternoon of the day before (from noon to 4 p.m.) and in the morning of the current day (from 9 a.m. to noon) to contribute to a result produced daily at noon.

The delay parameter, when specified, ensures that the evaluation of the expression E valid at time t is actually computed at time $t + \delta$: in general, the results (both valid at time t) computed at time t and at time $t + \delta$ may differ as some required contents of the temporal relations may not be available at time t yet, or even because their contents may have been retroactively changed after t (and also tuples in the streaming tables might be inserted with a little delay with respect to their validity, e.g., to enforce the right timestamp order). Consider, for instance, our example of insider trading detection that, in order to produce one result per trading day, needs to compare the trading volumes evaluated using the streaming data valid on a 1-month window preceding the execution time and on a 1-day window following the execution time (to this purpose, it would be sufficient to delay the execution at the end of the day). However, it also needs to join such volumes with the news concerning the same stocks published within one week. Since we can assume relevant news are selected and inserted by human analysts, we should also consider that they are likely inserted into the `NEWS` table retroactively, with the delay of some days with respect to their publication date. Hence, we should reasonably allow for a delay of, say, 10-15 days in the execution of the CQ in order to have all the relevant news available, otherwise the result of the join would always be empty and the insider trading cases could not be detected.

Notice that, when the streaming tables involved are defined at a finer time *granularity* than sl (e.g., sl="1 hour" but new tuples can be inserted into the streaming tables at every second), the values of E are usually defined for many more time points than required by the query. Hence, the sampling operator can be used to exactly specify the query execution timepoints of interest. This is also the reason for which we said in Sec. 2 that non-temporal tables (appearing in the expression E) are considered *virtually* converted into temporal tables

that contain an infinite number of tuples: only the tuples timestamped with one of the timepoints of interest have actually to be generated.

According to the generally accepted definition of continuous query semantics [16], we define the semantics of a continuous query $Q^{c,\delta}_{hp,sl,a}$ denoted by an algebraic expression E in the continuous temporal algebra $\mathcal{CTA}$ to be equal to the sampling of E at the time points specified by the slide parameter sl and the alignment parameter a.

▶ **Definition 14** (Semantics of continuous queries over streaming and standard tables)**.** Let $E_{\mathcal{CTA}} = E_{\mathcal{TA}}(\alpha_1(\omega_1(S_1)), \ldots, \alpha_n(\omega_n(S_n)), R_1, \ldots, R_m)$ be an algebraic expression in $\mathcal{CTA}$ over n streaming tables $S_1, \ldots, S_n$, with $n \geq 1$, and m temporal tables $R_1, \ldots, R_m$, with $m \geq 0$, where $E_{\mathcal{TA}}$ is an expression in $\mathcal{TA}$, and α_i and ω_i, $i = 1, \ldots, n$, are window aggregation/flattening and sliding window operators, respectively.

The result at time t of the continuous query $Q^{c,\delta}_{hp,sl,a}$ with historical period parameter hp, slide parameter sl, alignment parameter a and delay parameter δ, expressed by $E_{\mathcal{CTA}}$ is the streaming table with historical period hp given by the sampling $\xi^{t,\delta}_{hp,sl,a}(E_{\mathcal{CTA}})$ of $E_{\mathcal{CTA}}$ at the time points specified by sl with alignment a, and evaluation delayed by δ, until t.

It is worth noting that, as the CQ semantics is founded on the sampling operator, we actually implement a CQ on-demand semantics that produce at the execution query time successive query evaluations at past query points, thus in a delayed mode (also when δ=0). Moreover, the semantics of joining streaming tables and temporal and standard tables as specified in the algebraic expression $E_{\mathcal{TA}}$ refers to the standard temporal semantics. In this way, we implement different kinds of joining semantics according to the involved tables. For instance, when temporal tables are involved, the joining results will be temporally consistent according to the required time points in the past. When, instead, standard tables are involved, the joining semantics allow users to "interoperate" current data with past streamed data.

5 Example reprise

In order to express the query solving our insider trading detection example problem presented in Sec. 1, the following partitioned sliding windows need to be defined:

$$W_D = W^{\text{time},\texttt{STOCK}}_{[0,1\texttt{day}]}(\texttt{OPTION\_TRADES}), \quad W_M = W^{\text{time},\texttt{STOCK}}_{[1\texttt{month},0]}(\texttt{OPTION\_TRADES})$$

The former is a 1-day wide sliding windows following the evaluation time, partitioned according to the `STOCK` values. The latter is a 1-month wide sliding windows preceding the evaluation time, partitioned according to the `STOCK` values. Then, for our convenience, we can define the following streaming table expressions involving partitioned window aggregation:

$$S_D = \texttt{STOCK}\Theta_{\texttt{SUM(CONTRACTS)}}(W_D), \quad S_M = \texttt{STOCK}\Theta_{\texttt{SUM(CONTRACTS)}}(W_M)$$

(implying the computation of the total number of contracts executed, for each timepoint, in the time windows W_D and W_M, respectively) to be used in the definition of the algebraic expression that follows:

$$E = [\varepsilon_U(S_D) \bowtie_{S_D.\texttt{STOCK}=\texttt{NEWS.STOCK} \wedge (\texttt{NEWS}.U - S_D.U) < 1\texttt{week}} \pi_{\texttt{STOCK},U}(\varepsilon_U(\texttt{NEWS}))] \bowtie^T_{S_D.\texttt{STOCK}=S_M.\texttt{STOCK} \wedge S_D.\texttt{SUM(CONTRACTS)} > 10 \cdot S_M.\texttt{SUM(CONTRACTS)}/30} S_M$$

The square brackets enclose a non-temporal join that embodies the non-sequenced part of the query, which is necessary to interoperate the tuples valid at time t of the streaming table S_D with the tuples of the `NEWS` relation valid at a time no later than 1 week with respect to t. To this purpose, the timestamping attributes of S_D and `NEWS` have to be made explicit

via the extension operator ε to be referenced in the join predicate. Such join predicate also contains a conjunct imposing the equality of the `STOCK` value. The result is a streaming table whose timestamps are derived from S_D (the timestamp of `NEWS` has been projected out before executing the non-temporal join) and correspond to the execution time of the sliding window W_D. Once the expression in square brackets has been computed, a temporal join (on the streaming table timestamps) can be executed between the non-temporal join outcome and the streaming table S_M. The join predicate involves the equality of the `STOCK` value and the relationship between the results of the `SUM(CONTRACTS)` partitioned aggregates computed with Θ over S_M and S_D. In particular, the former (representing a daily volume) has to be greater than 10 times the value of the latter (representing the total volume computed over the preceding month) divided by 30 (yielding the average daily volume computed over the preceding month), in order to trigger an insider trading investigation. The continuous query result is finally given by the expression

$$\xi^{\mathrm{now},15\mathtt{day}}_{1\mathtt{day},0}(\pi^{T}_{\mathtt{STOCK}}(E)),$$

where the required sampling operator has been added, which causes the expression E to be evaluated (with a delay of 15 days to allow the `NEWS` table to be populated with relevant data) to produce a result each day at midnight containing the stocks suspect of illegal insider trading activity occurred on that day. The result is a temporal table with schema $(\mathtt{STOCK}|T)$.

6 Translating CQs into OTQs (with Implementation on the Horizon)

In this section we propose a translation of the continuous temporal model presented so far into a new temporal model where continuous queries are transformed into temporal one-time queries. Furthermore, whereas the continuous model has been defined as an *abstract temporal model* (point-based), the new model is intended to be a *concrete temporal model* (interval-based) [12] amenable to implementation. In particular, the new temporal model can be implemented on a traditional relational DBMS following similar directions as presented in [13]. In fact, our final aim is to build comprehensive support for the continuous temporal model through a mixed stratum/built-in approach that relies on the full potentialities of an industrial-strength relational engine, extended with novel functionalities.

For the intended translation, the source algebra is therefore $\mathcal{CTA}$ and the target algebra is the standard temporal algebra $\mathcal{TA}$ that includes the operators shown in Tab. 1 but made to work on relations employing interval-timestamping, according to an extended sequenced semantics [13], in order to enforce *snapshot equivalence*. Although working on an interval-based concrete temporal model, the target algebra represents indeed a point-based query language (in the sense of [28]) and, thus, its implementation on a traditional DBMS does not require enforcement of *change preservation* (e.g., via adjustment, alignment and scaling techniques as proposed in [13]). For ease of presentation, hereinafter, with a little abuse of notation, when we need to distinguish the same concept at the two different levels, we will use the superscript $\mathcal{CTA}$ to denote tables and algebraic operators in the continuous temporal model and the superscript $\mathcal{TA}$ to denote the corresponding concepts in the target model.

The main issue for our goal is to mimic in a static context the behavior of $\mathcal{CTA}$ windowing operators, which are evaluated on user-specified time intervals and operate on the contents of the involved streaming tables at the evaluation instants. To this end, we first translate each streaming table $S^{\mathcal{CTA}}$ with schema $S(X|T)$ at the continuous level into an interval-based streaming table $S^{\mathcal{TA}}$ with schema $S(X,T|T')$, where the event occurrence time T associated to tuples is made explicit and T' is an implicit interval attribute that records tuple validity, that is $[st,\infty]$, where st is the time when the tuple s enters the system (without

Substreaming operator
$\text{Sub}_{[t_1,t_2]}(S)^{\mathcal{TA}} := \sigma^T_{t_1 \leq S.T \leq t_2}(S)$
Sliding window operators
${}^{tset}w^{\text{time}}_{[\omega_1,\omega_2]}(S)^{\mathcal{TA}} := \bigcup_{t \in tset} \tau_t(\text{Sub}_{[t-\omega_1,t+\omega_2]}(S)^{\mathcal{TA}})$ ${}^{tset}w^{\text{count}}_{[n_1,n_2]}(S)^{\mathcal{TA}} := \bigcup_{t \in tset} \tau_t(\{s \mid s \in S' = \text{Sub}_{[t_1,t_2]}(S),$ $\lvert\text{Sub}_{[t_1,t]}(S')^{\mathcal{TA}}\rvert = n_1, \lvert\text{Sub}_{[t,t_2]}(S')^{\mathcal{TA}}\rvert = n_2\})$
Window flattening operators
$\varphi(S)^{\mathcal{TA}} := S$ $\Phi({}^{tset}W^{\text{time},B}_{[\omega_1,\omega_2]}(S))^{\mathcal{TA}} := {}^{tset}w^{\text{time}}_{[\omega_1,\omega_2]}(S)^{\mathcal{TA}}$ $\Phi({}^{tset}W^{\text{count},B}_{[n_1,n_2]}(S))^{\mathcal{TA}} := {}^{tset}w^{\text{count}}_{[n_1,n_2]}(S)^{\mathcal{TA}}$
Window aggregation operators
$\vartheta_F(S)^{\mathcal{TA}} := ({}_{\emptyset}\vartheta^T_F(S))^{\mathcal{TA}}$ ${}_B\Theta_F({}^{tset}W^{\text{time},B}_{[\omega_1,\omega_2]}(S))^{\mathcal{TA}} := ({}_B\vartheta^T_F({}^{tset}w^{\text{time}}_{[\omega_1,\omega_2]}(S)))^{\mathcal{TA}}$ ${}_B\Theta_F({}^{tset}W^{\text{count},B}_{[n_1,n_2]}(S))^{\mathcal{TA}} := ({}_B\vartheta^T_F({}^{tset}w^{\text{count}}_{[n_1,n_2]}(S)))^{\mathcal{TA}}$

Table 3 Semantics of the windowing operators at the target level, $\mathcal{TA}$.

transaction-time support, it is worth noting that *st* can be approximated with $s(T)$). Each temporal relation $R(X|T)$ in $\mathcal{CTA}$ is translated into a relation $R(X|T')$ in $\mathcal{TA}$, by *coalescing* the timestamps of *value-equivalent* tuples in $R^{\mathcal{CTA}}$ into maximal intervals to be used as timestamps in $R^{\mathcal{TA}}$. Non temporal relations are converted into temporal relations whose tuples are timestamped with a $[0, \infty]$ validity interval in $\mathcal{TA}$.

Then, in Tab. 3 we define the semantics of the continuous operators introduced in Subsection 4.2 defined through $\mathcal{TA}$ operators[2]. Notice that, unlike their counterpart at the $\mathcal{CTA}$ level, sliding window operators at the $\mathcal{TA}$ level require a set of time instants *tset* to be evaluated and the flattening operator $\Phi^{\mathcal{TA}}$ simply undoes the effects of partitioning. It is worth stressing that, thanks to the translation rules of Tab. 3, any legal $\mathcal{CTA}$ expression can be evaluated via $\mathcal{TA}$ operators working on streaming tables only. In particular, there is no need for implementing (sets of) streaming tables of streaming tables as formally introduced in the definitions of $\mathcal{CTA}$ operators in Sec. 4.

Finally, we define the sampling operator at the $\mathcal{TA}$ level and show that the results of the two sampling operators, the one defined at the $\mathcal{CTA}$ level and the other one defined at the $\mathcal{TA}$ level, are equivalent (i.e., they provide the same results for the same continuous query).

▶ **Definition 15** (Sampling Operator$^{\mathcal{TA}}$)**.** At execution time t, the evaluation delayed by δ of a continuous query $E = E_{\mathcal{TA}}(\alpha_1(\omega_1(S_1)), \ldots, \alpha_n(\omega_n(S_n)), R_1, \ldots, R_m) \in \mathcal{CTA}$, with an historical parameter *hp*, slide parameter *sl* and alignment parameter *a*, at the $\mathcal{TA}$ level is defined by the sampling operator $\xi^{\mathcal{TA}} : \mathcal{TA} \times \mathcal{T} \times \mathcal{I}^4 \to \mathcal{S}^{\mathcal{TA}}$ as follows:

$$\xi^{t,\delta}_{hp,sl,a}(E)^{\mathcal{TA}} := E_{\mathcal{TA}}((\alpha_1({}^{tset}\omega_1(S_1^{t+\delta})))^{\mathcal{TA}}, \ldots, (\alpha_n({}^{tset}\omega_n(S_n^{t+\delta})))^{\mathcal{TA}}, R_1^{t+\delta}, \ldots, R_m^{t+\delta})$$

where *tset* is the evaluation time instant set: $tset = \{t' \mid t' \leq t \wedge t' = (\lceil \frac{t-hp-a}{sl} \rceil + i) \cdot sl + a \text{ for some } i \in \mathbb{N}\}$.

[2] To be rigorous, the definition of $w^{\text{count}}_{[n_1,n_2]}$, involving constraints on the cardinality of the results of Sub, is not expressible with $\mathcal{TA}$ operators only. However, we can augment $\mathcal{TA}$ with an operator directly evaluating $w^{\text{count}}_{[n_1,n_2]}$, which can be easily and efficiently implemented by exploiting the ordering of timestamps in a streaming table.

▶ **Theorem 16.** *Given the continuous query* $E = E_{\mathcal{TA}}(\alpha_1(\omega_1(S_1)), \ldots, \alpha_n(\omega_n(S_n)), R_1, \ldots, R_m) \in \mathcal{CTA}$ *, with slide parameter* sl *and alignment parameter* a*, then, for each execution time* t *with delay* δ*:*

$$\xi^{t,\delta}_{sl,a}(E)^{\mathcal{CTA}} = \xi^{t,\delta}_{sl,a}(E)^{\mathcal{TA}}$$

Proof. For the sake of simplicity, we assume $\delta = 0$ and replace each R^t and S^t in the operator semantics with R and S, respectively (the proof can be straightforwardly adapted to the case when $\delta > 0$). First, notice that:

$$\begin{aligned}\xi^{t,\delta}_{sl,a}(E)^{\mathcal{CTA}} &= (\bigcup_{i=0}^{k:t_k \leq t} \tau_{t_i}(E_{\mathcal{TA}}(\alpha_1(\omega_1(S_1)), \ldots, \alpha_n(\omega_n(S_n)), R_1, \ldots, R_m)))^{\mathcal{CTA}} \\ &= (E_{\mathcal{TA}}(\bigcup_{i=0}^{k:t_k \leq t} \tau_{t_i}(\alpha_1(\omega_1(S_1))), \ldots, \bigcup_{i=0}^{k:t_k \leq t} \tau_{t_i}(\alpha_n(\omega_n(S_n))), R_1, \ldots, R_m))^{\mathcal{CTA}}\end{aligned}$$

Therefore, if we show that $\bigcup_{i=0}^{k:t_k \leq t} \tau_{t_i}(\alpha_j(\omega_j(S_j))^{\mathcal{CTA}}) = (\alpha_j({}^{tset}\omega_j(S_j)))^{\mathcal{TA}}$, then

$$\begin{aligned}&E_{\mathcal{TA}}(\bigcup_{i=0}^{k:t_k \leq t} \tau_{t_i}(\alpha_1(\omega_1(S_1))), \ldots, \bigcup_{i=0}^{k:t_k \leq t} \tau_{t_i}(\alpha_n(\omega_n(S_n))), R_1, \ldots, R_m))^{\mathcal{CTA}} \\ &= E_{\mathcal{TA}}((\alpha_1({}^{tset}\omega_1(S_1)))^{\mathcal{TA}}, \ldots, (\alpha_n({}^{tset}\omega_n(S_n)))^{\mathcal{TA}}, R_1, \ldots, R_m)\end{aligned}$$

and $\xi^{t,\delta}_{sl,a}(E)^{\mathcal{CTA}} = \xi^{t,\delta}_{sl,a}(E)^{\mathcal{TA}}$.

To this end, as far as $\alpha_j(\omega_j(S_j))$ is concerned, all possible operator combinations should be considered. For the sake of brevity, in the following we will consider only the case when $\alpha_j = \vartheta_F$ and $\omega_j = w^{\text{time}}_{[\omega_1,\omega_2]}$, but all the other combinations can be managed in a similar way. Given $\vartheta_F(w^{\text{time}}_{[\omega_1,\omega_2]}(S_j))$, in the following we will show that $s \in \bigcup_{i=0}^{k:t_k \leq t} \tau_{t_i}(\vartheta_F(w^{\text{time}}_{[\omega_1,\omega_2]}(S_j))^{\mathcal{CTA}})$ iff $s \in \xi^{t,\delta}_{sl,a}(E)^{\mathcal{TA}}$. Let $s \in \bigcup_{i=0}^{k:t_k \leq t} \tau_{t_i}(\vartheta_F(w^{\text{time}}_{[\omega_1,\omega_2]}(S_j))^{\mathcal{CTA}})$, then $s = (X, t_i) \in \tau_{t_i}(\vartheta_F(w^{\text{time}}_{[\omega_1,\omega_2]}(S_j))^{\mathcal{CTA}})$ for some i. Being $s \in \tau_{t_i}(\vartheta_F(w)^{\mathcal{CTA}})$, where $w = w^{\text{time}}_{[\omega_1,\omega_2]}(S_j)$, then $s \in \tau_{t_i}(\vartheta_F(w)^{\mathcal{CTA}})$ iff $t = (X, t_i)$, where $X = (f_1(S), \ldots, f_h(S))$ and $(S, \tau) \in w$. This means that $S = \text{Sub}_{[t_i-\omega_1, t_i+\omega_2]}(S_j)^{\mathcal{CTA}} = \{(s, \tau) \,|\, (s, \tau) \in S_j, (t_i - \omega_1) \leq \tau \leq (t_i + \omega_2)\}$. Hence, $(s, \tau) \in \text{Sub}_{[t_i-\omega_1, t_i+\omega_2]}(S_j)^{\mathcal{CTA}}$ iff $(s, \tau, t_i) \in \tau_{t_i}(\text{Sub}_{[t_i-\omega_1, t_i+\omega_2]}(S_j)^{\mathcal{TA}})$. As $t_i \in tset$, it follows that:

$$(s, \tau, t_i) \in \bigcup_{t \in tset} \tau_t(\text{Sub}_{[t-\omega_1, t+\omega_2]}(S_j)^{\mathcal{TA}}) =^{tset} w^{\text{time}}_{[\omega_1,\omega_2]}(S_j)^{\mathcal{TA}}$$

From $S = \tau_{t_i}(\text{Sub}_{[t_i-\omega_1, t_i+\omega_2]}(S_j)^{\mathcal{TA}})$ and $X = (f_1(S), \ldots, f_h(S))$, it follows that $(X, t_i) \in {}_{\emptyset}\vartheta^T_F(S_j)^{\mathcal{TA}}$, which is equivalent to say that $(X, t_i) \in (\vartheta_F(w^{\text{time}}_{[\omega_1,\omega_2]}(S_j))^{\mathcal{TA}}$. ◀

Therefore, thanks to the above theorem, we can safely translate each continuous query in $\mathcal{CTA}$ into an equivalent expression in $\mathcal{TA}$ and execute it on a static relational engine. In fact, the above theorem ensures that the semantics of execution is preserved.

7 Related works and concluding remarks

DSMSs [2, 4] natively support CQs over continuous unbounded streams of data according to windows where only the most recent data is retained. In CQL [4] and SyncSQL [15] streams are transformed into instantaneous/syncronized relations that are manipulated through relational operators, and then transformed back to streams. In this paper we proved that it

is possible to exploit the full potential of a native representation of temporal data to query streaming data seamlessly, thus overcoming the transformation overhead of stream-relation-stream approaches like [4, 15]. In line with this approach, recent research proposals extend traditional DBMSs' query model and language towards streaming query capabilities [11, 20]. However, these works present extensions to SQL through query examples and do not offer a formal algebraic framework for a clear specification of query operators.

With regard to algebras for querying streaming data, in [19] snapshot-reducible algebraic operators are implemented on ad-hoc data structures for state maintenance under a time-interval approach. This approach does not integrate with the theoretical and practical solutions proposed for the development of a robust temporal database technology, including [13] which presents a proposal for the implementation of a standard temporal algebra in an off-the-shelf DBMS, supporting a sequenced semantics that guarantees extended snapshot reducibility. As to $\mathcal{CTA}$ operators, we proved this fundamental requirement by showing how $\mathcal{CTA}$ expressions can be translated into equivalent expressions in the temporal algebra $\mathcal{TA}$.

An additional note concerns the semantics of ad-hoc proposals of temporal operators that often proves to be ambiguous as to timestamp management. A consensus is not shared among existing approaches. For instance, the timestamps of tuples resulting from a windowed join can be either the minimum of the two original timestamp values [2, 24], or the most recent one [5], or the time instant at which the join is executed [6]. Operators in $\mathcal{TA}$ undergo a precise and commonly adopted temporal semantics [13]. Further, in [4, 19] windowing operators overwrite the original timestamp of tuples with a new timestamp corresponding to the window evaluation time instant. $\mathcal{CTA}$ operators maintain instead both this information and the original tuple timestamp, thus not losing relevant information.

A further property featured by $\mathcal{CTA}$ is the capability of defining "forward" windows, thus opening to the possibility of evaluating queries (possibly in an approximated way) by referring tuples that will be observed after a given time instance, as proposed for SQL:2011 [29] (e.g., SQLStream Blaze[3] considers them in its query syntax specifications but does not provide an implementation). Moreover, both in SQL:2011 and proposed stream query languages, sliding windows can only be used, via aggregation operators, to produce results in the target list of a query, whereas $\mathcal{CTA}$ allows us to use them everywhere (e.g., in a selection predicate as in our running example). To the authors' knowledge, these features are not covered by any existing approach dealing with streaming data.

From a system perspective, existing stream processing frameworks (e.g., Apache Flink [8] and Samza [22]) provide SQL extensions to deal with streaming data but they do not expose a clear and unambiguous semantics of query operators through an algebra definition. In general, DSMSs and stream processing frameworks [8, 22] do not support queries involving both streaming and relational data changing over time, since their data and query models do not include temporal semantics and versioning. On the other hand, much work has been devoted to extending DBMSs towards a flexible and efficient management of temporal data [25]. However, quite surprisingly considering the temporal nature of streaming data, no built-in streaming functionalities are provided in these systems.

Following the first step provided by the streaming table concept introduced in [7], in this paper we presented a temporal algebra extended with windowing and aggregation operators supporting both OTQs and CQs on streaming, standard and temporal relational data. In our future work, we plan to explore algebraic optimization issues and indexing techniques to efficiently support the implementation of $\mathcal{CTA}$ operators in a temporal DBMS.

[3] http://sqlstream.com

References

1 Insider Trading. `https://www.sec.gov/fast-answers/answersinsiderhtm.html`.

2 Daniel J. Abadi et. al. Aurora: a new model and architecture for data stream management. *VLDB J.*, 12(2):120–139, 2003.

3 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

4 Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL continuous query language: semantic foundations and query execution. *VLDB J.*, 15(2):121–142, 2006.

5 Ahmed M. Ayad and Jeffrey F. Naughton. Static Optimization of Conjunctive Queries with Sliding Windows over Infinite Streams. In *Proc of ACM SIGMOD*, pages 419–430, 2004.

6 Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and Issues in Data Stream Systems. In *Proc. of ACM PODS*, pages 1–16, 2002.

7 Luca Carafoli, Federica Mandreoli, Riccardo Martoglia, and Wilma Penzo. Streaming tables: Native support to streaming data in dbmss. *IEEE Trans. on Systems, Man, and Cybernetics: Systems*, pages 1–15, 2017.

8 Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink™: Stream and batch processing in a single engine. *IEEE Data Eng. Bull.*, 38(4):28–38, 2015.

9 Cristina De Castro, Fabio Grandi, and Maria Rita Scalas. Semantic interoperability of multitemporal relational databases. In *Proceedings of the 12th International Conference on the Entity-Relationship Approach*, pages 463–474. Springer-Verlag, 1993.

10 Ugur Cetintemel, Jiang Du, Tim Kraska, Samuel Madden, David Maier, John Meehan, Andrew Pavlo, Michael Stonebraker, Erik Sutherland, Nesime Tatbul, Kristin Tufte, Hao Wang, and Stanley Zdonik. S-Store: A Streaming NewSQL System for Big Velocity Applications. *Proc. VLDB*, 7(13):1633–1636, August 2014.

11 Qiming Chen and Meichun Hsu. Cut-and-rewind: Extending query engine for continuous stream analytics. In A. Hameurlain, J. Kueng, R. Wagner, A. Cuzzocrea, and U. Dayal, editors, *TLDKS XXI*, volume 9260 of *LNCS*, pages 94–114. 2015.

12 Jan Chomicki. Temporal query languages: A survey. In *Proceedings of the First International Conference on Temporal Logic*, pages 506–534. Springer-Verlag, 1994.

13 Anton Dignös, Michael H. Böhlen, Johann Gamper, and Christian S. Jensen. Extending the kernel of a relational dbms with comprehensive support for sequenced temporal queries. *ACM Trans. Database Syst.*, 41(4):26:1–26:46, 2016.

14 Steve Donoho. Early detection of insider trading in option markets. In *Proc. of the ACM KDD*, pages 420–429. ACM, 2004.

15 Thanaa M. Ghanem, Ahmed K. Elmagarmid, Per-ake Larson, and Walid G. Aref. Supporting Views in Data Stream Management Systems. *ACM Trans. Database Syst.*, 35(1):1, 2010.

16 Lukasz Golab and M. Tamer Özsu. Update-Pattern-Aware Modeling and Processing of Continuous Queries. In *Proc. of ACM SIGMOD*, pages 658–669, 2005.

17 Fabio Grandi. Temporal interoperability in Multi+Temporal databases. *J. Database Manag.*, 9(1):14–23, 1998.

18 Christian S. Jensen, Curtis E. Dyreson, Michael H. Böhlen, James Clifford, Ramez Elmasri, Shashi K. Gadia, Fabio Grandi, Patrick J. Hayes, Sushil Jajodia, Wolfgang Käfer, Nick Kline, Nikos A. Lorentzos, Yannis G. Mitsopoulos, Angelo Montanari, Daniel A. Nonen, Elisa Peressi, Barbara Pernici, John F. Roddick, Nandlal L. Sarda, Maria Rita Scalas, Arie Segev, Richard T. Snodgrass, Michael D. Soo, Abdullah Uz Tansel, Paolo Tiberio, and Gio Wiederhold. The consensus glossary of temporal database concepts - february 1998 version. In S. Sripada O. Etzion, S Jajodia, editor, *Temporal Databases: Research and Practice*, pages 367–405. Springer-Verlag, 1998.

19 Jürgen Krämer and Bernhard Seeger. Semantics and Implementation of Continuous Sliding Window Queries over Data Streams. *ACM Trans. Database Syst.*, 34(1):4, 2009.

20 Nikolay Laptev, Barzan Mozafari, Hani Mousavi, Hetal Thakkar, Haixun Wang, Kai Zeng, and Carlo Zaniolo. *Extending Relational Query Languages for Data Streams*, pages 361–386. Springer Berlin Heidelberg, 2016.

21 Erietta Liarou, Stratos Idreos, Stefan Manegold, and Martin Kersten. Enhanced stream processing in a dbms kernel. In *Proc. of EDBT*, pages 501–512, 2013.

22 Yi Pan Milinda Pathirage, Julian Hyde and Beth Plale. SamzaSQL: Scalable fast data management with streaming SQL. In *Proc. of IEEE IPDP Symposium Workshops*, pages 1627–1636. IEEE Computer Society, 2016.

23 Emanuele Panigati, Fabio A. Schreiber, and Carlo Zaniolo. *Data Streams and Data Stream Management Systems and Languages*, pages 93–111. Springer, 2015.

24 Kostas Patroumpas and Timos Sellis. Window specification over data streams. In *Current Trends in Database Technology - EDBT 2006 Workshops*, pages 445–464, 2006.

25 Dušan Petkovic. *Temporal Data in Relational Database Systems: A Comparison*, pages 13–23. Springer, 2016.

26 Richard T. Snodgrass, editor. *The TSQL2 Temporal Query Language*. Kluwer, 1995.

27 Utkarsh Srivastava and Jennifer Widom. Flexible Time Management in Data Stream Systems. In *Proc. of ACM PODS*, pages 263–274, 2004.

28 David Toman. Point vs. interval-based query languages for temporal databases. In *Proc. of ACM PODS*, pages 58–67. ACM Press, 1996.

29 Fred Zemke. What's new in SQL:2011. *SIGMOD Rec.*, 41(1):67–73, April 2012.