This is a pre print version of the following article:

Unleashing the power of querying streaming data in a temporal database world: A relational algebra approach / Grandi, F., Mandreoli, F., Martoglia, R., Penzo, W.. - In: INFORMATION SYSTEMS. - ISSN 0306-4379. - 103:(2022), pp. 1-25. [10.1016/j.is.2021.101872]

Terms of use:

The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

18/11/2026 13:58

(Article begins on next page)

Unleashing the Power of Querying Streaming Data in a Temporal Database World: A Relational Algebra Approach

Fabio Grandi[a], Federica Mandreoli[b], Riccardo Martoglia[b,*], Wilma Penzo[a]

[a] *DISI, University of Bologna, viale Risorgimento, 2, I-40136 Bologna, Italy*
[b] *FIM, University of Modena and Reggio Emilia, via Campi, 213/b, I-41125 Modena, Italy*

Abstract

Modern data-intensive applications have to manage very large quantities of continuously streaming data in combination with standard relational data and need advanced query capabilities that involve the combination of continuous queries (CQs) and one-time queries (OTQs) to satisfy complex analytics requirements, including verification of temporal conditions and data versioning specifications.

Current approaches bridging standard database and stream data management solutions provide only limited declarative query capabilities that do not fully exploit the potential of a real synergic approach given by the full integration of both worlds.

In this paper, we lay the foundations of a query language unifying streaming, temporal, and standard relational querying capabilities under a common denominator that proves to be more powerful than simply the union of its parts. In particular, we introduce a comprehensive algebraic framework that founds on the full acknowledgment of the inherently temporal nature of streaming data to support CQs and OTQs seamlessly on streaming, temporal, and standard relational tables. This is done by: (a) introducing a temporal algebra $\mathcal{TA}^{\star}$ for OTQ specification; (b) extending $\mathcal{TA}^{\star}$ to a continuous temporal algebra $\mathcal{CTA}$ for CQ specification; (c) providing the semantics of OTQs and CQs in the defined framework; (d) showing how OTQs and CQs expressed in the $\mathcal{CTA}$ formalism can be translated into relational algebraic expressions amenable to implementation on top of a traditional DBMS.

Keywords: Continuous queries, Data streams, Relational algebra, Temporal DB

*Corresponding author.
Email addresses: fabio.grandi@unibo.it (Fabio Grandi), federica.mandreoli@unimo.it (Federica Mandreoli), riccardo.martoglia@unimo.it (Riccardo Martoglia), wilma.penzo@unibo.it (Wilma Penzo)

1. Introduction

Modern data-intensive applications, including advanced surveillance (e.g., financial market enforcement), monitoring applications in smart cities and healthcare, network applications (e.g., intrusion detection), often require an increasingly wider range of data management capabilities in order to be fully supported. On the one hand, they need to manage very large quantities of continuously streaming data in combination with standard relational data. Moreover, they also need to store the whole incoming streams as historical data in order to make them fully persistent and available to future analysis. On the other hand, they need advanced query capabilities that involve the combination of continuous queries (CQs) and one-time queries (OTQs) to satisfy complex analytics requirements, including temporal conditions and data versioning specifications.

For instance, in the context of database forensics, we can consider applications that need to review on-line activities and to ex-post evaluate user responsibility. For example, one application could verify that a financial broker did not sell the stocks of a client, although the bearish indicators and the stop-loss signal triggered on his/her computer screen suggested to do it. The analysis of the broker's past actions taken on the basis of the real-time recommendations given by a stream processing system could be straightforwardly effected by means of a continuous query that joins data belonging to different temporal versions.

Nowadays, in order to cope with such advanced requirements, existing applications implement ad-hoc solutions based on the bridge-building between stream data management and standard database capabilities, in a hybrid fashion [1, 2]. However, they have to rely on currently available systems that provide query languages offering only limited declarative capabilities in comparison with a typical (temporal) DBMS approach. In fact, query languages provided by big data processing engines and stream processing frameworks (e.g.,[3, 4, 5, 6]), as well as by streaming extensions to traditional DBMSs (e.g.,[7, 8, 9]), do not support data versioning and temporal querying. On the other hand, the reference temporal query language TSQL2 [10] and its evolutions are not designed to deal with streams and to support continuous queries. Thus, in order to satisfy complex analytics requirements, database queries have to be flanked with application-specific procedures, resulting in this way in an overall mixed declarative-procedural query approach.

In this paper, we lay the foundations of a query language that unifies streaming, temporal, and standard relational querying capabilities under a common denominator that proves to be more powerful than simply the union of its parts. As a matter of fact, the query language we envision could allow for the specification of queries never supported before in a completely declarative way. To this end, we propose a comprehensive relational algebra, named $\mathcal{CTA}$ (Continuous Temporal Algebra). While a general consensus on the formal semantics of query operators and, in general, of queries has been accomplished for streaming [11], for temporal [10], and for standard relational data [12], separately, this definitely lacks in the unified scenario we envision. From a practical viewpoint, it is important to have an equivalent algebra for a query language or calculus because

query evaluation is typically implemented by mapping a query into an algebraic expression. Moreover, manipulation of equivalent algebraic expressions provides the basis for query optimization. The paper provides a proof of this practical implication by showing how $\mathcal{CTA}$ queries can be translated into a concrete temporal model amenable to implementation on top of a traditional relational DBMS.

Our proposal relies on the full acknowledgement of the inherently temporal nature of streaming data. As a preliminary step, in [13] we introduced a new kind of temporal table, named *streaming table*, as the essential abstraction for representing streaming data into a DBMS in a native way. In a streaming table, a stream of data entering the system is kept historical for a user-defined period. Streaming tables can be straightforwardly involved in OTQs and CQs, possibly together with standard relational tables. In pursuing the goal of setting up our algebraic framework in a concrete temporal relational environment, we rely on the work presented in [14] where operators of a standard temporal algebra $\mathcal{TA}$ supporting a sequenced semantics [15] are proved to be implementable in an off-the-shelf non-temporal DBMS.

The proposed algebraic framework is the outcome of the following main contributions:

- We extend the standard temporal algebra $\mathcal{TA}$ by adding non-temporal operators, which are native in every traditional DBMS. As we show in the paper, this is aimed at supporting more expressive temporal queries that involve data belonging to different temporal snapshots. We denote such extended temporal algebra as $\mathcal{TA}^\star$; OTQs can be formulated as $\mathcal{TA}^\star$ expressions over standard, temporal and streaming tables.

- We introduce a continuous temporal algebra supporting CQs seamlessly on streaming, temporal, and standard relational tables. The algebra we propose, denoted ad $\mathcal{CTA}$, extends $\mathcal{TA}^\star$ with windowing operators; CQs, can be formulated as $\mathcal{CTA}$ expressions that are $\mathcal{TA}^\star$ expressions over standard and temporal tables and *windowing expressions* on streaming tables; A wide range of windowing operators are provided to this end, including partitioned versions that mimic the SQL group-by mechanism;

- We formally define the semantics of OTQs and CQs. OTQs' semantics relies on the well-founded $\mathcal{TA}^\star$ definition, and is based on the sequenced semantics of $\mathcal{TA}$ operators and on the standard semantics of non-temporal operators. CQs' semantics relies on the adoption of a *sampling operator* that evaluates $\mathcal{CTA}$ expressions at the *time points* specified through a set of evaluation parameters. In this way, we introduce a sort of *on-demand semantics*, where CQs are executed when query results are needed and required data are available. Windowing expressions in CQs, built as combinations of windowing operators, take streaming tables as input and produce streaming tables storing data snapshots at subsequent validity time points;

- We extend the work in [14] a step further, by proposing the translation of $\mathcal{CTA}$ expressions to an algebraic form derived from $\mathcal{TA}^\star$ and amenable to implementation. In this way, OTQs and CQs can be optimized and executed on top of a traditional DBMS (with a kernel extended as shown in [14]).

A preliminary version of the proposed framework has been presented in [16]. This article extends [16] in various directions. The most relevant one is the proposal of the temporal algebra $\mathcal{TA}^\star$. The paper provides a complete syntax specification of $\mathcal{TA}^\star$ and details how $\mathcal{TA}^\star$ overcomes the severe limitations of previously proposed temporal relational algebras, including $\mathcal{TA}$. The introduction of $\mathcal{TA}^\star$ has required a substantial rework of the framework as now $\mathcal{CTA}$ relies on $\mathcal{TA}^\star$.

The paper is organized as follows. In Sec. 2 we review the notion of streaming table and provide the motivating example referenced throughout the paper. The extended temporal algebra $\mathcal{TA}^\star$ and the continuous temporal algebra $\mathcal{CTA}$ are introduced in Secs. 3 and 4, respectively. Section 5 shows how $\mathcal{CTA}$ expressions can be supported through the introduction of a sampling operator and Sec. 6 expresses the reference example in $\mathcal{CTA}$. In Sec. 7 we present a correct translation of continuous queries formulated as $\mathcal{CTA}$ expressions into temporal queries in $\mathcal{TA}^\star$, making it possible to instantiate the proposed framework in the context of a standard temporal RDBMS. The paper is complemented by Sec. 8 comparing our contribution with the state of the art, and by Sec. 9 where we draw our conclusions and outline future research directions. Finally, a comprehensive discussion of the limitations of temporal algebra $\mathcal{TA}$ with sequenced semantics and of the extensions needed to support any kind of temporal queries is carried out in Appendix A and Appendix B.

2. Preliminaries

We start briefly recalling some basic concepts on temporal and streaming data that are preparatory to next Sections. Then, we introduce a motivating example to give a flavour of the powerful queries that could be formulated on a mix of relational, temporal and streaming data.

2.1. Basics on temporal and streaming data

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 [13].

First of all, we assume a discrete, ordered and unbounded time domain $\mathcal{T} = \{0, 1, 2, \ldots, now, \ldots, \infty\}$ composed of *chronons* [18], where 0 stands for the earliest time and *now* the current time. A chronon, as defined in [18], 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 [19], 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 when they have to be interoperated with temporal tables and streaming tables by using a suitable temporal conversion map [20]. In particular, we assume a non-temporal table R can be virtually converted to a temporal table as follows $(R)^T = \{(r, \tau) \,|\, r \in R, \tau \in \mathcal{T}\}$. The fact that a "valid always" timestamping is added to non-temporal data could look as an arbitrary choice. For example, a more conservative choice of assigning timestamps $\tau \in [now, \infty]$ timestamp, where *now* is the execution time of a temporal query referencing that data, was made in [20]. On the other hand, an assumption universally made in the temporal database field (e.g., which is at the base of interval-timestamping) is the *step function continuity assumption* [21], stating that temporal data after insertion keep their constant value until they are changed or deleted. Considering data found *now* in a non-temporal relation, we can assume its validity starts from a time t in the past corresponding to its insertion time. Since we have no information about the insertion time t, we make an unrestricted choice by setting $t = 0$. In this way, constant values stored in a non-temporal table can be interoperated with any data contained in (each snapshot of) a temporal table via a temporal algebra operator.

As far as streams are concerned, we adopt the definition of continuous data stream (or simply stream) provided in [11], that is a potentially infinite sequence of timestamped relational tuples having a fixed schema.

A *streaming table* is a special kind of relational table, first introduced in [13], that, unlike a standard relational table, is subject to continuous insertions of streaming data. In a streaming table, 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 modelled as an event table [10, 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* [10, 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. For each tuple $u \in S$, the expression $u.\nu$ denotes the tuple insertion number in S (i.e., $u.\nu = n$ iff u is the n^{th} inserted tuple). The content of S_{hp} at any time*

instant t, that is S^t_{hp}, is a set of tuples such that:

- *for each tuple $u = (s, \tau)$ the timestamp $T(u) = \tau$ satisfies $\tau \geq \max(t - hp, 0)$; if positive, $t - hp$ represents the chronon preceding t by a time span of hp chronons.*
- *for any u_1 and u_2 in S^t_{hp}, $T(u_1) < T(u_2)$ implies $u_1.\nu < u_2.\nu$, that is u_1 was inserted before u_2.*

In practice, in order to implement this insertion semantics, systems cope with out-of-order and skewed inputs. Interested readers can refer to [22] for an in-depth discussion of this aspect. In this paper we assume an input manager that guarantees in-order tuple arrival. Notice that, in order not to burden the notation, we neither represent ν as an explicit nor an implicit attribute in tuples, but we assume that its values can always be determined for each tuple stored in a streaming table (e.g., via a system-managed `ROW_NUMBER` function). Still for ease of notation, whenever possible in the following, we will use S in place of S_{hp}.

In the following, let $\mathcal{R}$ be the set of all temporal tables (also including the non-temporal ones) 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. Indeed, 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.

2.2. Motivating Example

As an application example in the context of financial market surveillance, we consider tracking of *insider trading* activities. According to the US Securities and Exchange Commission (SEC), insider trading consists of the buying or selling of a security while in possession of nonpublic material information [23]. Examples of such material information include, for instance, a company being up/downgraded by a rating agency, unexpected revisions to earning results or projections, mergers and acquisitions news. 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 [24], 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 stock option trading data are available (cf., owing to their financial leverage, options are usually the elective way to capitalize on insider information) through the streaming table `OPTION_TRADES`, 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

OPTION_TRADES

OPTION	STOCK	CLASS	STRIKE	EXPIR	CONTRACTS	T(sec)
Opt245	Tesla	Put	300	Jan 2018	480	45
Opt237	Apple	Put	180	Jan 2018	360	44
Opt245	Tesla	Put	300	Jan 2018	350	42
Opt127	Google	Call	1100	Dec 2017	150	35
Opt127	Google	Call	1100	Dec 2017	200	33
Opt666	Apple	Put	120	Nov 2017	15000	28

NEWS

STOCK	TYPE	SOURCE	T(day)
Tesla	Merger and Acquisition	Wall Street Journal	4
Apple	Q4 Financial Results	Financial Times	4
Google	Fitch Rating	Financial Times	3
Tesla	Dividend	Tesla Investor Relations	5
Google	Capital Increase	Wall Street Journal	3

Figure 1: Running example tables: The OPTION_TRADES streaming table (top) and the NEWS temporal table (bottom)

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 containing information about the publication date (i.e., the implicit attribute T defined with a granularity of one day[1]), the mentioned stock, the news type and source.

Sample tables for the application example are shown in Fig. 1. For instance, the first tuple of OPTION_TRADES represents a trade of 480 contracts involving a put option on the Tesla stock. Such an option, bought to bet against the Tesla stock, will pay off if the stock at the end of January 2021 is below the strike price of $300. We assume the sample transactions shown in Fig. 1 have been all executed on Day 0 (e.g., the trade in the first tuple has been closed at the second 45 of Day 0). On the other hand, the first tuple of table NEWS represents a merger and acquisition news concerning the Tesla company published on the Wall Street Journal on Day 4. 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. For instance, this could be the case of the Apple stock option Opt666 recorded in table OPTION_TRADES, which shows a seemingly very high volume and precedes by approximately 4 days the

[1]Notice that timestamps of the OPTION_TRADES and NEWS table are shown with a different granularity in Fig. 1 for clarity of the example, but timestamps of all the tables in a database are assumed to be stored with a common base granularity (e.g., of one second).

publication of some bad financial results of Apple in Q4 (second tuple of table NEWS).

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. Furthermore, such a temporal join cannot be expressed in a purely snapshot-reducible [25] temporal relational algebra, as it requires interoperating data belonging to different temporal snapshots (see Appendix A).

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

In OTQs a streaming table is dealt with as a standard (event) table and it undergoes queries expressed in a temporal relational algebra.

3.1. Supporting basic OTQs with the temporal algebra $\mathcal{TA}$

The starting point of our proposal is the temporal algebra $\mathcal{TA}$ presented in [14], equipped with the following primitive 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 semantics of the $\mathcal{TA}$ 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 (e.g., COUNT, SUM, AVG, ...), 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. The grouping operator performs aggregation by groups. More precisely, it partitions the tuples of a temporal table into groups according to their common values on attributes in B, and it applies the aggregate functions in $F \subseteq \mathcal{F}$ to each group. Finally, notice that, differently form the one defined in [14], 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.

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

Table 1: $\mathcal{TA}$ operator semantics

The definition of these operators is borrowed from [14, Sec. 2 and Appendix A] and, thus, $\mathcal{TA}$ supports a sequenced semantics in terms of extended snapshot reducibility. In particular, an extend operator ε_U is available, which can be used to copy the timestamp T to the additional explicit attribute U. As defined in [14], *extended* snapshot reducibility refers precisely to the support of queries that reference in such a way the original timestamps. However, the original $\mathcal{TA}$ operators with sequenced semantics allow the use of timestamp data to a very limited extent and the resulting query language has a poor expressive power (e.g., it does not allow to express queries that combine data belonging to different database snapshots). Therefore, as discussed in Appendix A, in order to support a wider range of interesting temporal queries, we have to further extend it as shown in the Subsection that follows.

3.2. Supporting powerful OTQs with the extended temporal algebra $\mathcal{TA}^\star$

Expressive temporal queries, like those that can be expressed with the TSQL2 language [10], have an overall structure that must be evaluated with a sequenced semantics and parts that must be evaluated with a non-sequenced semantics. The former include an implicit temporal join (with intersection of the valid

$E_{\mathcal{TA}^\star}$	$\rightarrow$	Q^T
Q^T	$\rightarrow$	R^T, $\mathsf{op}_1^T(Q^T)$, $R^T\,\mathsf{op}_2^T\,Q^T$, $Q^T \times Q$, $Q \times Q^T$
op_1^T	$\rightarrow$	π_X^T, σ_F^T, ${}_B\vartheta_F^T$, ε_U, τ_t
op_2^T	$\rightarrow$	$\times^T$, $\cup^T$, $-^T$
R^T	$\rightarrow$	temporal table, streaming table,
		$(Q)^T$ (non-temporal expression virtually converted to temporal)
Q	$\rightarrow$	R, $\mathsf{op}_1(Q)$, $Q\,\mathsf{op}_2\,R$,
		$\pi_Y(Q^T)$ (Y is a subset of the explicit attributes of Q^T)
op_1	$\rightarrow$	π_X, σ_F, ${}_B\vartheta_F$
op_2	$\rightarrow$	$\times$, $\cup$, $-$
R	$\rightarrow$	non-temporal table

Table 2: Complete syntax of a $\mathcal{TA}^\star$ expression $E_{\mathcal{TA}^\star}$

timestamps) between all the relations appearing in the `SELECT` clause. The latter include the evaluation of temporal selection predicates, appearing in the WHERE clause, referencing data belonging to different snapshots at the same time, like the example query introduced in Sec. 6, which cannot be expressed using $\mathcal{TA}$ (cf., discussion in Appendix A).

The temporal query language ATSQL [25], which has been designed to supersede TSQL2, provided for separate modalities of temporal execution following the sequenced or the non-sequenced semantics through the use of statement modifiers `SEQ` and `NSEQ`, respectively, prefixing each query. In such a way, the advantages of a "hybrid" semantics (i.e., partly sequenced and partly non-sequenced) are lost: pure sequenced queries suffer from a very limited expressivity and pure non-sequenced queries waive the benefits of a sequenced execution, including the efficient support of interval-timestamping (and change preservation) with the implementation techniques presented in [14].

Hence, the extended temporal algebra $\mathcal{TA}^\star$ we propose is based on a hybrid semantics, in order to benefit from a combined approach. In fact, both modalities can be supported in the expression and during the execution of a query, provided that it is clear which parts of the language must be evaluated with a sequenced or non-sequenced semantics. In particular, $\mathcal{TA}^\star$ extends $\mathcal{TA}$ to the support of the non-sequenced part of queries by allowing the use of standard (i.e., non-temporal) algebra operators in subexpressions (e.g., to join data belonging to different snapshots). In such subexpressions, in order to correctly produce a temporal result, a streaming/temporal table can be combined by means of a non-temporal product operator (e.g., $\times$ or $\bowtie$) with a non-temporal table only (which can even be a streaming/temporal table projected on explicit attributes only by means of π). In such a case, the result inherits the timestamps from the streaming/temporal operand. On the other hand, non-temporal tables or non-temporal algebra subexpressions which are arguments of a $\mathcal{TA}$ operator are

always virtually converted to temporal before the application of the temporal operator (as shown in Sec. 2.1).

Further details on this choice of a hybrid approach and on its motivations can be found in Appendix A. Examples of legal use of non temporal operators in a $\mathcal{TA}^\star$ expression will be given in Sec. 6 and in Appendix A.

In practice, like the TSQL2 language, our proposed $\mathcal{TA}^\star$ provides a snapshot-reducible algebra template allowing for *islands* of non-reducibility inside. In particular, the production rules for any expression $E^\star_{\mathcal{TA}}$ in $\mathcal{TA}^\star$ are shown in Tab. 2. The first five production rules define the snapshot-reducible template, whereas the last four ones define the islands of non-reducibility. In particular, Q^T may contain subexpressions $Q^T \times Q$ or $Q \times Q^T$, which must produce a correct temporal relation (i.e., a relation with a single timestamp T) but contain at least one (as root of their syntax tree) non-temporal relational algebra operator for which the standard (i.e., non-sequenced) semantics applies, and may involve $\mathcal{TA}^\star$ subexpressions for which Definition 2 recursively applies.

Notice that the expression $\sigma^T_F(Q^T \times Q)$ is equivalent to $Q^T \bowtie_F Q$, as F only involves explicit attributes, which may also be the result of timestamp propagation via the extend operator: this latter is the pattern, also used in our examples, which allows one to express non-sequenced temporal join conditions between different database snapshots. For instance, let Q_1 and Q_2 be two aliases for the $\mathcal{TA}$ expression $\sigma^T_{\texttt{STOCK=Tesla}\wedge\texttt{Year}(U)=2018}(\varepsilon_U(\texttt{NEWS}))$ selecting the news published in 2018 about Tesla, then the $\mathcal{TA}^\star$ expression $\pi^T_{\texttt{SOURCE}}(Q_1 \bowtie_{Q_1.U+\texttt{1day}=Q_2.U} \pi_U(Q_2))$, where the (left) right operand of the non-temporal join is a (temporal) non-temporal relation, retrieves the sources where news about Tesla were published on two consecutive days in 2018. Using primitive operators only, the latter expression is clearly equivalent to $\pi^T_{\texttt{SOURCE}}(\sigma^T_{Q_1.U+\texttt{1day}=Q_2.U}(Q_1 \times \pi_U(Q_2)))$.

Notice that the ATSQL language can be used in a similar way by nesting non-sequenced statements into sequenced statements (e.g., to translate temporal logic expressions as shown in [26]); however, the resulting statements with possibly multiple nestings come out unnecessarily complex even for quite simple temporal SPJ queries, with respect to the TSQL2-like syntax we propose for $\mathcal{TA}^\star$ and for which a single `SELECT` statement suffices as shown in Sec. 6 and Appendix A. To the best of our knowledge, no temporal algebras which can be used with a hybrid semantics were formalized before $\mathcal{TA}^\star$.

In order to highlight its being an evolutionary step with respect to $\mathcal{TA}$ (for which the expression "extended snapshot reducibility" was coined), we say that $\mathcal{TA}^\star$ supports an *extended** *snapshot reducibility*.

In the following, we assume readers are familiar with the syntax and semantics of relational queries[2] 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 $\mathcal{TA}^\star$ expression $E_{\mathcal{TA}^\star} = Q^T(S_1, \ldots, S_n, R_1, \ldots, R_m, (Q_1)^T, \ldots, (Q_\ell)^T)$,*

[2]Interested readers can refer to [12] for an in-depth study.

over n streaming tables $S_1, \ldots, S_n$, with $n \geq 1$, m temporal tables $R_1, \ldots, R_m$, with $m \geq 0$, and ℓ non-temporal subexpressions $Q_1, \ldots, Q_\ell$, with $\ell \geq 0$, virtually converted to temporal, its semantics is the result of the semantics of the involved temporal operators for which extended* snapshot reducibility holds, that is for each $t \in \mathcal{T}$: $\tau_t(E_{\mathcal{TA}^\star}) = Q^T(\tau_t(S_1), \ldots, \tau_t(S_n), \tau_t(R_1), \ldots, \tau_t(R_m), \tau_t((Q_1)^T), \ldots, \tau_t((Q_\ell)^T))$.

The necessity of non-sequenced algebraic subexpressions for expressing *real temporal queries* and the design choices leading to the development of $\mathcal{TA}^\star$ are discussed and exemplified in Appendix A. From a theoretical point of view, notice that the possibility of accessing the timestamp values (via the extend operator) and of using them as if they were explicit attributes for all intents and without limitations (also via non-temporal algebra operators), makes our $\mathcal{TA}^\star$ language as expressive as the Temporal Relational Calculus (TRC, TS-FO in [27] or 2-FOL in [28]), which has been proven to be strictly more expressive than a language based on First-Order Temporal Logic (FOTL), with "implicit" access to time like the temporal algebras with snapshot evaluation [29], including that of [14]. For example, the TRC "snapshot-equality" query, which was proven in [27, 28] that cannot be expressed in FOTL, can be expressed in our extended temporal algebra as shown in Appendix B. These features are summarized by the Lemma that follows:

Lemma 1. *The extended* snapshot-reducible temporal algebra $\mathcal{TA}^\star$ has the same expressivity of the Temporal Relational Calculus and, thus, is strictly more expressive than a purely snapshot-reducible temporal algebra or an extended snapshot-reducible temporal algebra like $\mathcal{TA}$.*

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

A continuous query is a query that is issued once, and then logically runs continuously until terminated by the user. Any streaming table S referenced in a continuous query must be accessed through a sliding window expression $w(S)$ 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 [30], by generalizing their semantics to generate and operate on, possibly through aggregation, *sequences* of windows instead of single windows. Thus, $\mathcal{CTA}$ further extends the temporal algebra $\mathcal{TA}^\star$ with the introduction of such generalized windowing operators ($\mathcal{TA} \subset \mathcal{TA}^\star \subset \mathcal{CTA}$). The following definition introduces the notion of algebraic expression in the continuous temporal algebra $\mathcal{CTA}$.

Definition 3 ($\mathcal{CTA}$ expression). *An algebraic expression $E_{\mathcal{CTA}}$ in the continuous temporal algebra $\mathcal{CTA}$ is an expression $Q^T(w_1(S_1), \ldots, w_n(S_n), R_1, \ldots, R_m)$ over n streaming tables $S_1, \ldots, S_n$, with $n \geq 0$ and each S_i accessed through a corresponding sliding window expression w_i, and m temporal tables $R_1, \ldots, R_m$, with $m \geq 0$ (these may also include streaming tables used as temporal tables,*

that is without the interposition of sliding window operators, and non-temporal expressions virtually converted to temporal).

The complete syntax of a general $\mathcal{CTA}$ expression $E_{\mathcal{CTA}}$ is the same as in Table 2 (with $E_{\mathcal{CTA}}$ replacing $E_{\mathcal{TA}^\star}$ in the first place) augmented by the production rule $R^T \to w(S)$, where S is a streaming table.

Furthermore, in order to have a well-defined semantics of continuous queries, we must add to the $\mathcal{CTA}$ syntax the constraint that non-temporal Q subexpressions cannot contain sliding window expressions (they may contain anyway streaming tables, which are dealt with as temporal tables). This means to forbid that Q^T subexpressions appearing in $\pi_Y(Q^T)$ expressions (where Y is a subset of the explicit attributes of Q^T) may contain sliding window expressions $w(S)$.

4.1. $\mathcal{CTA}$ sliding window expressions

In order to support sliding window expressions (w) over streaming tables, $\mathcal{CTA}$ introduces two classes of operators: one includes *sliding window operators* (ω) and the other includes *window flattening and aggregation operators* (α). Sliding window expressions are then defined in a modular way as the composition of one of the former with one of the latter operators: $w = \alpha \circ \omega$. Both classes of operators include one *standard* and one *partitioned* version of each operator. Standard operators operate over streaming tables whereas partitioned operators operate over streaming table partitions, that is sets of streaming tables.

In the formalization of ω and α operators, in addition to the streaming table datatype $\mathcal{S}$, we will also make use of the following exotic datatypes:

set of streaming tables: $2^{\mathcal{S}} = \{S \,|\, S \in \mathcal{S}\}$

streaming table of streaming tables: $\mathcal{S}^* = \{(S, \tau) \,|\, S \in \mathcal{S}, \tau \in \mathcal{T}\}$

set of streaming tables of streaming tables: $2^{\mathcal{S}^*} = \{S^* \,|\, S^* \in \mathcal{S}^*\}$

We underline that these exotic datatypes are only used in definitions for the formalization of the semantics of the operators, but they are not part of the data model, as the output of any legal sliding window expression $w(S)$ is always a streaming table.

For the sake of clarity, we give an intuition of these two classes of operators that work in a complementary fashion. All sliding window operators ω generate a timestamped sequence of portions of the input streaming table according to a sliding window specification. The new timestamps assigned to each portion correspond to times at which the windows are evaluated (whereas each portion maintains as "temporal provenance" witnesses the original timestamps of the tuples selected from the input streaming table). Specifically, the output of base sliding window operators is formally a streaming table of streaming tables whereas the output of partitioned sliding window operators is formally a set of streaming tables of streaming tables. Window flattening operators and window aggregation operators α reduce the result of a sliding window operator to a streaming table, in any case.

In the following, we provide the formal definition and semantics of the operators involved in the specification of sliding window expressions. For ease of notation, we start by introducing some utility operators. The definition of $\mathcal{CTA}$ operators and the specification of their combinations in legal expressions follow.

4.2. Utility operators

The following operators provide useful transformations of streaming tables, which are indeed used for the definition of $\mathcal{CTA}$ operators.

Definition 4 (Time-Substreaming). *The time-substreaming operator* $\mathrm{Sub}^T : \mathcal{S} \times \mathcal{T}^2 \to \mathcal{S}$ *restricts a streaming table* $S \in \mathcal{S}$ *to only tuples such that their timestamp belongs to an interval* $[t_1, t_2]$*:*

$$\mathrm{Sub}^T_{[t_1,t_2]}(S) := \{u \,|\, u \in S \wedge t_1 \leq T(u) \leq t_2\}.$$

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

Definition 5 (Order-Substreaming). *The order-substreaming operator* $\mathrm{Sub}^\nu : \mathcal{S} \times \mathcal{T}^2 \to \mathcal{S}$ *restricts a streaming table* $S \in \mathcal{S}$ *to only tuples such that their insertion number belongs to an interval* $[\nu_1, \nu_2]$*:*

$$\mathrm{Sub}^\nu_{[\nu_1,\nu_2]}(S) := \{u \,|\, u \in S \wedge \nu_1 \leq u.\nu \leq \nu_2\}.$$

For instance, the expression $\mathrm{Sub}^\nu_{[201,\ 300]}(\texttt{OPTION_TRADES})$ builds from `OPTION_TRADES` a streaming table containing the 100 trades inserted after the first 200 ones.

Definition 6 (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*):*

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

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`.

4.3. Sliding window operators (w)

In accordance with commonly adopted definitions of sliding windows [30], sliding window operators in $\mathcal{CTA}$ are time-based and count-based. Standard sliding window operators apply to a streaming table S and formally generate a streaming table of streaming tables. Definitions of (backward and/or forward) standard sliding window operators are provided below.

Definition 7 (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:*

$$w^{\text{time}}_{[\omega_1,\omega_2]}(S) := \{(S', \tau) \,|\, \exists \tau \in \mathcal{T} \wedge S' = \text{Sub}^T_{[\tau-\omega_1,\tau+\omega_2]}(S)\}.$$

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 8 (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 temporally closest* $n_1 \geq 0$ *tuples valid before and the temporally closest* $n_2 \geq 0$ *tuples valid after the timestamps around which it is computed (the tuples valid at the timestamps around which it is computed are also included):*

$$w^{\text{count}}_{[n_1,n_2]}(S) := \{(S', \tau) \,|\, \exists \tau \in \mathcal{T} \wedge S' = \text{Sub}^{\nu}_{[\nu_1-n_1,\nu_2+n_2]}(S) \\ \wedge \nu_1 = \min\{u.\nu \,|\, u \in S \wedge T(u) = \tau\} \wedge \nu_2 = \max\{u.\nu \,|\, u \in S \wedge T(u) = \tau\}\}.$$

In the definition of (S', τ) above, the tuples $u \in S$ with $\nu_1 \leq u.\nu \leq \nu_2$ are the all and only tuples valid at time τ. 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, and one tuple immediately following τ.

The partitioned version of each sliding window operator applies to a streaming table S and formally 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 9 (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 10 (Count-based Partitioned Window). *The count-based partitioned sliding window operator* $W^{\text{count}} : \mathcal{S} \times 2^{\mathcal{A}} \times \mathbb{N}^2 \rightarrow 2^{\mathcal{S}^*}$ *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 temporally closest* $n_1 \geq 0$ *tuples valid before and the temporally 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*:*

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

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 and aggregation operators (α)

The definition of window flattening and aggregation operators is introduced in the following.

4.4.1. Window flattening operators

Window flattening operators allow for *normalizing* the output of a time-based or a count-based sliding window operator in $\mathcal{S}^*$ or $2^{\mathcal{S}^*}$, respectively, to a streaming table in $\mathcal{S}$. As for sliding window operators, the window flattening operator is introduced both in its standard and in its partitioned version.

Definition 11 (Window Flattening). *The window flattening operator* $\varphi : \mathcal{S}^* \rightarrow \mathcal{S}$ *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:*

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

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.

Definition 12 (Partitioned Window Flattening). *The partitioned window flattening operator* $\Phi : 2^{\mathcal{S}^*} \rightarrow \mathcal{S}$ *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:*

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

For instance, the expression $\Phi(W^{\text{count},\texttt{OPTION},\texttt{CLASS}}_{[4,0]}(\texttt{OPTION_TRADES}))$, 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.

It is worth noting that, for both flattening operators, 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.

4.4.2. 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 13 (Window Aggregation). *The sliding window aggregation operator $\theta : \mathcal{S}^* \times 2^{\mathcal{F}} \rightarrow \mathcal{S}$ 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:*

$$\theta_F(w) := \{(Z, \tau) \mid \exists (S, \tau) \in w \wedge Z = (f_1(S), \ldots, f_h(S))\}.$$

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

Definition 14 (Partitioned Window Aggregation). *The partitioned sliding window aggregation operator $\Theta : 2^{\mathcal{S}^*} \times 2^{\mathcal{A}} \times 2^{\mathcal{F}} \rightarrow \mathcal{S}$ 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:*

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

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 $\texttt{EXPIR}\Theta_{\texttt{COUNT(OPTION)}}(W^{\text{time},\texttt{EXPIR}}_{[0.5\text{hour},0.5\text{hour}]}(\texttt{OPTION_TRADES}))$, 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.

4.5. Legal sliding window expressions

α	ω	w
$\varphi \mid \theta$	$w^{\text{time}} \mid w^{\text{count}}$	$\varphi(w^{\text{time}}_{[\omega_1,\omega_2]}(S))$, $\varphi(w^{\text{count}}_{[n_1,n_2]}(S))$, $\theta_F(w^{\text{time}}_{[\omega_1,\omega_2]}(S))$, $\theta_F(w^{\text{count}}_{[n_1,n_2]}(S))$
$\Phi \mid \Theta$	$W^{\text{time}} \mid W^{\text{count}}$	$\Phi(W^{\text{time},B}_{[\omega_1,\omega_2]}(S))$, $\Phi(W^{\text{count},B}_{[n_1,n_2]}(S))$, ${}_B\Theta_F(W^{\text{time},B}_{[\omega_1,\omega_2]}(S))$, ${}_B\Theta_F(W^{\text{count},B}_{[n_1,n_2]}(S))$

Table 3: $\mathcal{CTA}$ operators combinations in legal sliding window expressions w

The modularity of α and ω operators is regulated by the following definition:

Definition 15 (Legal Sliding Window Expression). *A $\mathcal{CTA}$* legal sliding window expression $w(S)$ *over the streaming table S is of the form $\alpha(\omega(S))$ where standard w^{time} and w^{count} (resp., partitioned W^{time} and W^{count}) sliding window operators can only be combined with their standard φ and θ (resp., partitioned Φ and Θ) window flattening or window aggregation counterparts.*

The admitted combinations are shown in Table 3 (combinations of standard operators in the first row, of partitioned operators in the second row). These constraints ensure that the value of continuous expressions augmenting $\mathcal{TA}^\star$ is always a streaming table, so that the resulting continuous algebra $\mathcal{CTA}$ is closed with respect to (streaming and) temporal tables. Nevertheless, the definitions we provided in this section, also involving $2^{\mathcal{S}}$, $\mathcal{S}^*$ and $2^{\mathcal{S}^*}$ exotic datatypes, could be used as guidelines for a *native* implementation of the $\mathcal{CTA}$ operators, for which the materialization of sets of streaming tables or (sets of) streaming tables of streaming tables (which could be stored in a $\neg$1NF datastore, e.g., in XML or JSON format) could be used to represent intermediate results during the evaluation of $\alpha(\omega(S))$ window expressions. In our approach, materialization of such intermediate steps is unnecessary, as the translation that will be proposed in Sec. 7 maps any $\mathcal{CTA}$ expressions on $\mathcal{TA}^\star$ expressions to be executed on top of a temporal DBMS via manipulation of temporal relations only. From a semantic point of view, the use of streaming tables of streaming tables as intermediate results in the evaluation of $\alpha(\omega(S))$ expressions allowed the propagation of original timestamps of tuples making up a streaming table as provenance witnesses, while whole streaming tables are assigned by the ω operator a new global timestamp corresponding to the window evaluation time. Provenance witness timestamps are then used by the α operator to correctly

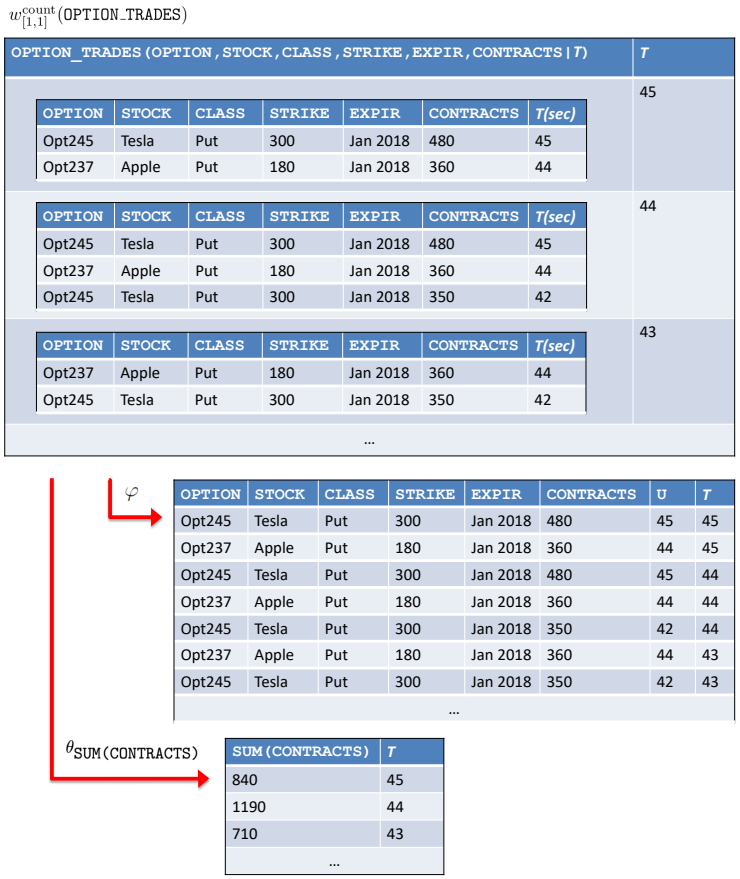


Figure 2: Examples of $\alpha(\omega(S))$ expressions: $\varphi(w^{\text{count}}_{[1,1]}(\texttt{OPTION_TRADES}))$ and $\theta_{\texttt{SUM(CONTRACTS)}}(w^{\text{count}}_{[1,1]}(\texttt{OPTION_TRADES}))$.

produce the desired final result in (1NF) streaming table format. In this respect, such provenance witness timestamps play a role similar to the lineage sets used in [14] to define change-preserving operators.

Example. Examples of how the results of $\alpha(\omega(S))$ expressions are conceptually constructed via the composition of a sliding window operator with a flattening or aggregation operator are shown in Fig. 2. The input streaming table S is OPTION_TRADES, the value of the intermediate sliding window expression $w^{\text{count}}_{[1,1]}(\texttt{OPTION_TRADES})$ is displayed on the top and the values of the final expressions $\varphi(w^{\text{count}}_{[1,1]}(\texttt{OPTION_TRADES}))$ and $\theta_{\texttt{SUM(CONTRACTS)}}(w^{\text{count}}_{[1,1]}(\texttt{OPTION_TRADES}))$ are displayed on the bottom of the figure.

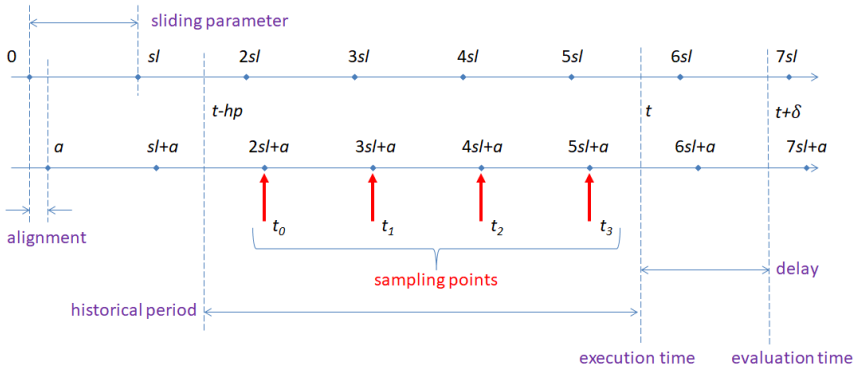


Figure 3: The sampling points defined at execution time t by the parameters hp, sl, a, δ.

5. 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., [11]), 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. 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`). 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 $t + \delta$. 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 [30].

Definition 16 (Sampling Operator). *At execution time t, the sampling operator $\xi : \mathcal{CTA} \times \mathcal{T} \times \mathcal{I}^4 \rightarrow \mathcal{S}$, with an historical period parameter hp, a sliding parameter sl, an alignment parameter a, causes the evaluation of the algebraic expression $E \in \mathcal{CTA}$ at time points $t_0, t_1, \ldots, t_k$ only, where $t_i = (\lceil \frac{t-hp-a}{sl} \rceil + i) \cdot sl + a$ and k is the largest natural number such that $t_k \leq t$. If a delay parameter δ is specified, it forces the evaluation of the expression E to be actually executed at time $t + \delta$:*

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

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 $w^{\text{time}}_{[1\text{day},0]}(\texttt{OPTION_TRADES})$ 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.

Fig. 3 shows how sampling points can be visually determined on the time axis according to the specified ξ parameters. In practice, sl and a define a sequence of time points with period sl shifted by a with respect to the time origin. The sampling points, highlighted by red arrows in the figure, are the points of such sequence that fall into the hp-wide interval preceding the execution time t.

When a delay value δ is specified, 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 [31], we define the semantics of a continuous query $Q^{c,\delta}_{hp,sl,a}$ denoted by an algebraic expression $E_{\mathcal{CTA}} \in \mathcal{CTA}$ to be equal to the sampling of $E_{\mathcal{CTA}}$ at the time points specified by the slide parameter sl and the alignment parameter a.

Definition 17 (Semantics of continuous queries over streaming and standard tables). *Let $E_{\mathcal{CTA}} = E_{\mathcal{TA}^\star}(w_1(S_1), \ldots, w_n(S_n), R_1, \ldots, R_m, Q_1, \ldots, Q_\ell)$ be an algebraic expression in $\mathcal{CTA}$ containing legal only sliding window expressions $w_i(S_i) = \alpha_i(\omega_i(S_i))$, with $1 \leq i \leq n$.*

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 produces 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 $E_{\mathcal{TA}^\star}$ 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 allows users to interoperate current data with past streamed data.

6. Example reprise

Let us come back to example the presented in Sec. and assume that *a posteriori* fraud analysis has to be performed over the last six months of trading data.

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

$$\begin{aligned} W_D &= W^{\text{time},\texttt{STOCK}}_{[0,\texttt{1day}]}(\texttt{OPTION_TRADES}), \\ W_M &= W^{\text{time},\texttt{STOCK}}_{[\texttt{1month},0]}(\texttt{OPTION_TRADES}) \end{aligned}$$

The former is a 1-day wide sliding window following the evaluation time, partitioned according to the `STOCK` values. The latter is a 1-month wide sliding window 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:

$$\begin{aligned} S_D &= \texttt{STOCK}\Theta_{\texttt{SUM(CONTRACTS)}}(W_D), \\ S_M &= \texttt{STOCK}\Theta_{\texttt{SUM(CONTRACTS)}}(W_M) \end{aligned}$$

(implying the computation of the total number of contracts executed, for each timepoint, in the time windows W_D and W_M, respectively). For instance, considering the sample data of our running example in Fig. 1, we can assume S_M computes an average monthly volume of around 30,000 contracts per each option, which yields an average daily volume of 1,000. Hence, S_D would compute an anomalous high total of Apple contracts (i.e., larger than 15,000) for the considered day (Day 0). In order to detect it, S_D and S_M are used in the definition of the algebraic expression that follows:

$$\begin{aligned} E \;=\; & [\varepsilon_U(S_D) \bowtie_{S_D.\texttt{STOCK}=\texttt{NEWS.STOCK} \wedge 0<(\texttt{NEWS}.U-S_D.U)<1\texttt{week}} \pi_{\texttt{STOCK},U}(\varepsilon_U(\texttt{NEWS}))] \\ & \bowtie^T_{S_D.\texttt{STOCK}=S_M.\texttt{STOCK} \wedge S_D.\texttt{SUM(CONTRACTS)}>10 \cdot S_M.\texttt{SUM(CONTRACTS)}/30} S_M \end{aligned}$$

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 within 1 week after t. To this purpose, the timestamping attributes of S_D and NEWS have to be made explicit via the extension operator ε_U 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. Notice that the expression within square brackets is a correct $\mathcal{TA}^\star$ expression, thanks to the equivalence $\sigma^T_F(Q^T \times Q) \equiv Q^T \bowtie_F Q$: in fact, with respect to this equivalence, the sliding window expression $\varepsilon_U(S_D)$ plays the role of Q^T, whereas $\pi_{\texttt{STOCK},U}(\varepsilon_U(\texttt{NEWS}))$ plays the role of the non-temporal subexpression Q, which is actually the only non-temporal subexpression appearing in E (hence, also the constraint ruling that in $\mathcal{TA}^\star$ a sliding window expression cannot appear within a non-temporal subexpression Q as defined in Tab. 2 is respected).

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

$$\xi^{\text{now},15\texttt{day}}_{6\texttt{month},1\texttt{day},0}(\pi^T_{\texttt{STOCK}}(E)),$$

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) over a past period of six months 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 $(\texttt{STOCK}|T)$.

In the case of our sample data (Fig. 1), the result would contain a tuple relating to "Apple" valid on Day 0, for which an anomalous contract volume has been detected via S_D and S_M and joined with the news of the presumably negative financial results (NEWS table) published 4 days after the anomalous transactions.

The continuous query expression $\xi^{\text{now},15\text{day}}_{6\text{month},1\text{day},0}(\pi^{T}_{\texttt{STOCK}}(E))$ above could be given the SQL-like syntax that follows:

```
SELECT DAY_VOL.STOCK, DAY_VOL.T
FROM OPTION_TRADES OVER DAY_WIN AS DAY_VOL,
     OPTION_TRADES OVER MON_WIN AS MON_VOL,
     NEWS
WHERE DAY_VOL.STOCK = MON_VOL.STOCK
  AND DAY_VOL.SUM(CONTRACTS) > 10*MON_VOL.SUM(CONTRACTS)/30
  AND DAY_VOL.STOCK = NEWS.STOCK
  AND NEWS.T > DAY_VOL.T
  AND NEWS.T < DAY_VOL.T + INTERVAL 1 WEEK
WINDOW DAY_WIN AS (PARTITION BY STOCK RANGE 1 DAY FOLLOWING),
       MON_WIN AS (PARTITION BY STOCK RANGE 1 MONTH PRECEDING)
SAMPLE INTERVAL 1 DAY ALIGN TO START DELAY 15 DAY
HISTORICAL PERIOD 6 MONTH
```

In this sample query, we basically adopted a window declaration syntax quite similar to that proposed for other streaming SQL extensions (e.g., SQLStream [32]). The main difference is that, whereas in previous proposals window expressions could only be used in the SELECT clause to produce query results, we also allow them in the WHERE clause together with flattening/aggregate operators (i.e., implementing $\alpha(\omega(S))$ expressions) in order to produce window-based data selection. In practice, as it is one of the qualifying features of our $\mathcal{CTA}$ algebra, we propose to also add this feature to a SQL query language extension for streaming tables, although allowing aggregates in the WHERE clause could look like a "stretch" of the SQL standard syntax. However, once window expressions have been constructed, aggregates computed on them can play the role of flat table attributes (e.g, see Fig. 2) and used for selection as plain attributes in the WHERE clause. Obviously, aggregates in the WHERE clause could only be used as attributes of window expressions. On the other hand, the translation of such a SQL query into a $\mathcal{CTA}$ expression is straightforward (see also Appendix A).

7. 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) [19] amenable to implementation. In particular, the new temporal model can be implemented on a traditional relational DBMS following similar directions as presented in [14]. 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 extended temporal algebra $\mathcal{TA}^\star$ presented in Sec. 3 but made to work on relations employing interval-timestamping, according to an extended sequenced semantics [14], in order to enforce *snapshot equivalence* also in the presence of subexpressions supporting a non-sequenced semantics. Although working on an interval-based concrete temporal model, the target algebra represents indeed a point-based query language (in the sense of [33]) 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 [14]). 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 $\mathcal{TA}^\star$ 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 $S^{\mathcal{TA}^\star}$ 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 transaction-time support, it is worth noting that st can be approximated with $s.T$). Each temporal relation with schema $R(X|T)$ in $\mathcal{CTA}$ is translated into a relation with schema $R(X|T')$ in $\mathcal{TA}^\star$, by *coalescing* the timestamps of *value-equivalent* tuples in $R^{\mathcal{CTA}}$ into maximal intervals to be used as timestamps in $R^{\mathcal{TA}^\star}$. Non temporal relations are converted into temporal relations whose tuples are timestamped with a $[0, \infty]$ validity interval in $\mathcal{TA}^\star$.

Then, in Tab. 4 we present the semantics of the continuous operators introduced in Subsection 4.1 defined through $\mathcal{TA}$ operators, to be used as translation rules from the source $\mathcal{CTA}$ to the target $\mathcal{TA}^\star$ language.

Notice that, unlike their counterpart at the $\mathcal{CTA}$ level, sliding window operators at the $\mathcal{TA}^\star$ level require a set of time instants $tset$ to be evaluated and the flattening operator $\Phi^{\mathcal{TA}}$ simply undoes the effects of partitioning. Being $\mathcal{TA} \subset \mathcal{TA}^\star$, with the translation rules of Tab. 4 converting sliding window expressions into $\mathcal{TA}$, any legal $\mathcal{CTA}$ expression can be translated into $\mathcal{TA}^\star$. Moreover, it is worth stressing that, in this way, any legal $\mathcal{CTA}$ sliding window expression can be evaluated via $\mathcal{TA}$ operators working on streaming tables only.

Sliding window operators
${}^{tset}w^{\text{time}}_{[\omega_1,\omega_2]}(S)^{\mathcal{TA}} := \bigcup_{t\in tset} \tau_t(\sigma^T_{t-\omega_1\leq S.T\leq t+\omega_2}(S))$
${}^{tset}w^{\text{count}}_{[n_1,n_2]}(S)^{\mathcal{TA}} := \bigcup_{t\in tset} \tau_t(S \ltimes^T_{\texttt{MIN}(\nu)-n_1\leq S.\nu\leq \texttt{MAX}(\nu)+n_2} \emptyset\vartheta^T_{\texttt{MIN}(\nu),\texttt{MAX}(\nu)} \sigma^T_{S.T=t}(S))$
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
$\theta_F(S)^{\mathcal{TA}} := (\emptyset\vartheta^T_F(S))^{\mathcal{TA}}$
${}_B\Theta_F({}^{tset}W^{\text{time},B}_{[\omega_1,\omega_2]}(S))^{\mathcal{TA}} := ({}_B\vartheta^T_F({}^{tset}w^{\text{time}}_{[\omega_1,\omega_2]}(S)))^{\mathcal{TA}}$
${}_B\Theta_F({}^{tset}W^{\text{count},B}_{[n_1,n_2]}(S))^{\mathcal{TA}} := ({}_B\vartheta^T_F({}^{tset}w^{\text{count}}_{[n_1,n_2]}(S)))^{\mathcal{TA}}$

Table 4: Translation of windowing operators into $\mathcal{TA}$

In particular, there is no need for implementing sets of streaming tables or (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 $\mathcal{TA}^\star$ 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 $\mathcal{TA}^\star$ level, are equivalent (i.e., they provide the same results for the same continuous query).

Definition 18 (Sampling Operator$^{\mathcal{TA}^\star}$). *At execution time t, the evaluation delayed by δ of an algebraic expression $E = E_{\mathcal{TA}^\star}(\alpha_1(\omega_1(S_1)),\ldots,\alpha_n(\omega_n(S_n)),R_1,\ldots,R_m) \in \mathcal{CTA}$, where R_j, with $1\leq j\leq m$, can be a temporal table or a streaming table used as a temporal table (i.e., without the interposition of sliding window operators) or a non-temporal $\mathcal{TA}^\star$ subexpression virtually converted to temporal, with an historical parameter hp, slide parameter sl and alignment parameter a, at the $\mathcal{TA}^\star$ level is defined by the sampling operator $\xi^{\mathcal{TA}^\star} : \mathcal{TA}^\star\times\mathcal{T}\times\mathcal{I}^4\to\mathcal{S}^{\mathcal{TA}^\star}$ as follows:*

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

where tset is the evaluation time instant set: $tset = \{t' \mid t'\leq t \wedge t' = (\lceil\frac{t-hp-a}{sl}\rceil + i)\cdot sl + a$ for some $i\in\mathbb{N}\}$ and ${}^{tset}R_j^{t+\delta} = \bigcup_{t_i\in tset}\tau_{t_i}(R_j^{t+\delta})$, with $1\leq j\leq m$.

The following Theorem shows the correctness of the translation. It is worth noting that correctness is shown for $\mathcal{CTA}$ algebraic expressions that include legal sliding window expressions w only. We recall that any legal sliding window expression $w(S)$ is in the form $\alpha(\omega(S))$, according to the combinations in Tab. 3, and always outputs a streaming table ($\mathcal{CTA}$ is closed). Therefore, the equivalence

of the two sampling operators, the one at $\mathcal{CTA}$ level and the other one at $\mathcal{TA}^\star$ level, concerns with the semantics of legal sliding window expressions $\alpha(\omega(S))$ at $\mathcal{CTA}$ level and the semantics of the corresponding expressions shown in Tab. 4.

Theorem 1. *Given the continuous query* $E = E_{\mathcal{TA}^\star}(\alpha_1(\omega_1(S_1)), \ldots, \alpha_n(\omega_n(S_n)), R_1, \ldots, R_m) \in \mathcal{CTA}$ *(where* R_j*, with* $1 \leq j \leq m$*, can be a temporal table or a streaming table* S *used as a temporal table, that is without the interposition of sliding window operators, or a non-temporal* $\mathcal{TA}^\star$ *subexpression virtually converted to temporal), with slide parameter* sl *and alignment parameter* a*, then, for each execution time* t *with delay* δ*:*

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

where $\xi^{\mathcal{CTA}}$ is the sampling operator defined for $\mathcal{CTA}$ in Def. 16.

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, we can observe that:

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

where ${}^{tset}R_j = \bigcup_{t_i\in tset} \tau_{t_i}(R_j) = \bigcup_{i=0}^{max\{k\in\mathbb{N}|t_k\leq t\}} \tau_{t_i}(R_j)$ $(1 \leq j \leq m)$.

Hence, recalling that non-temporal subexpressions (which may appear among R_js) cannot contain sliding window expressions and, thus, $(R_j)^{\mathcal{CTA}} = (R_j)^{\mathcal{TA}^\star} = R_j$ $(1 \leq j \leq m)$, if we show that $\bigcup_{i=0}^{max\{k\in\mathbb{N}|t_k\leq t\}} \tau_{t_i}(\alpha_j(\omega_j(S_j))^{\mathcal{CTA}}) = (\alpha_j({}^{tset}\omega_j(S_j)))^{\mathcal{TA}}$, then we can write:

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

And, thus, the proof of the theorem follows: $\xi^{t,\delta}_{sl,a}(E)^{\mathcal{CTA}} = \xi^{t,\delta}_{sl,a}(E)^{\mathcal{TA}^\star}$.

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 start from the case when $\alpha_j = \theta_F$ and $\omega_j = w^{\text{time}}_{[\omega_1,\omega_2]}$. Given $\theta_F(w^{\text{time}}_{[\omega_1,\omega_2]}(S_j))$, in the following we will show that $s \in \bigcup_{i=0}^{max\{k\in\mathbb{N}|t_k\leq t\}} \tau_{t_i}(\theta_F(w^{\text{time}}_{[\omega_1,\omega_2]}(S_j))^{\mathcal{CTA}})$ iff $s \in (\theta_F({}^{tset}w^{\text{time}}_{[\omega_1,\omega_2]}(S_j))^{\mathcal{TA}}$. Let $s \in \bigcup_{i=0}^{k:t_k\leq t} \tau_{t_i}(\theta_F(w^{\text{time}}_{[\omega_1,\omega_2]}(S_j))^{\mathcal{CTA}})$, then $s = (X, t_i) \in \tau_{t_i}(\theta_F(w^{\text{time}}_{[\omega_1,\omega_2]}(S_j))^{\mathcal{CTA}})$ for some i. Being $s = (X, t_i) \in \tau_{t_i}(\theta_F(w)^{\mathcal{CTA}})$, where $w = w^{\text{time}}_{[\omega_1,\omega_2]}(S_j))$, then, according to the $\theta_F(w)$ semantics, $s \in \tau_{t_i}(\theta_F(w)^{\mathcal{CTA}})$ iff $X = (f_1(S), \ldots, f_h(S))$ and $(S, t_i) \in w$. This means that $S = \text{Sub}^T_{[t_i-\omega_1,t_i+\omega_2]}(S_j)^{\mathcal{CTA}} = \{(s, \tau) \,|\, (s, \tau) \in S_j, (t_i - \omega_1) \leq \tau \leq (t_i + \omega_2)\}$.

Hence, $(s,\tau) \in \mathrm{Sub}^T_{[t_i-\omega_1,t_i+\omega_2]}(S_j)^{\mathcal{CTA}}$ iff $(s,\tau,t_i) \in \tau_{t_i}(\mathrm{Sub}^T_{[t_i-\omega_1,t_i+\omega_2]}(S_j)^{\mathcal{TA}})$. As $t_i \in tset$, it follows that:

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

From $S' = \tau_{t_i}(\mathrm{Sub}^T_{[t_i-\omega_1,t_i+\omega_2]}(S_j)^{\mathcal{TA}})$ 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 $(X',t_i) \in (\theta_F(w^{\mathrm{time}}_{[\omega_1,\omega_2]}(S_j))^{\mathcal{TA}}$. Being $X = X'$ then $s \in \bigcup_{i=0}^{max\{k\in\mathbb{N}|t_k\le t\}} \tau_{t_i}(\theta_F(w^{\mathrm{time}}_{[\omega_1,\omega_2]}(S_j))^{\mathcal{CTA}})$ iff $s \in (\theta_F({}^{tset}w^{\mathrm{time}}_{[\omega_1,\omega_2]}(S_j))^{\mathcal{TA}}$.

When $\alpha_j = \varphi$, X and X' definitions change but the equivalence still holds. Instead, when $\omega_j = w^{\mathrm{count}}_{[n_1,n_2]}$, the temporal windows in Sub^T change in $[t_1,t_2]$ but it still holds that $(s,\tau) \in \mathrm{Sub}^T_{[t_1,t_2]}(S_j)^{\mathcal{CTA}}$ iff $(s,\tau,t_i) \in \tau_{t_i}(\mathrm{Sub}^T_{[t_1,t_2]}(S_j)^{\mathcal{TA}})$. Finally, the proof for the partitioned operators can be built in a similar way. □

Therefore, thanks to the above theorem, we can safely translate each continuous query in $\mathcal{CTA}$ into an equivalent expression in $\mathcal{TA}^\star$ and execute it on a temporal database or even on a traditional relational engine (with an extended kernel supporting the execution of $\mathcal{TA}$ expressions as shown in [14]). In fact, the above theorem ensures that the semantics of execution is preserved.

8. Related work

In the following, we will provide a general overview of the status of current data management systems that only partially tackle the requirements we are targeting in this paper. Then we will discuss our contribution with respect to related works, specifically focusing on algebras for querying streaming and temporal data.

8.1. Query languages and algebras for querying streaming and/or temporal data

Many works on DSMSs exist in literature, many of them including interesting theoretical frameworks on algebras for querying streaming data. DSMSs (among them, Stream Mill [34], STREAM [11], Aurora [35], TelegraphCQ [36], NiagaraCQ [37], Gigascope [38]) natively support CQs over continuous unbounded streams of data according to windows where only the most recent data is retained [39]. This kind of research is mainly focused on performances of stream data management and querying rather than semantics. Some works specifically focus on semantics but in a very different context: for instance [40] addresses the problem of synopsis construction with the goal of keeping in the system the smallest amount of data possible w.r.t. query requirements, while our assumption is that data never leaves the system. Generally, (e.g., in CQL [11] and SyncSQL [41]) streams are transformed into instantaneous/synchronized relations that are manipulated through relational operators, and then transformed back to streams. Under a more general perspective, these approaches do not actually deal with both streaming and traditional relational data in a unified model.

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 [11, 41], as well as of recent proposals building on the same assumptions and focusing on out-of-order data management [42].

Given the above considerations, some works charge DSMSs with short-sightedness as to treating stream processing distinct from traditional data processing [7, 8, 9]. Thus, recent research proposals extend traditional DBMSs' query model and language towards streaming query capabilities [8, 43]. 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. The only paper offering a fully native representation of streaming data in a DBMS is [13].

The work in [30] presents a comprehensive description of window types (time-based and tuple-based sliding, landmark, partitioned, and tumbling windows) that can be specified over data streams and an algebra for continuous query specification. However, the algebraic operators introduced are tightly bound to the window types, thus resulting in windowed operators that absorb windows in their definition. In our work instead we propose a sharp distinction between algebraic operators, belonging to $\mathcal{TA}^{\star}$, and windows, resulting from the application of the window operators defined in $\mathcal{CTA}$. Such a twofold view leads to two main advantages: to take advantage of the well-founded temporal algebraic framework offered by $\mathcal{TA}^{\star}$, and more flexibility as to algebraic optimization.

8.2. Data-intensive systems and frameworks

From a system perspective, a great amount of software frameworks for the management of large volumes of data has flourished in recent years. However, none of them offer a full-fledged answer to the challenging data management objectives we pointed out in the introduction and that motivated this work.

DSMSs do not support data persistency and/or temporal versioning: the common assumption is "all past tuples cannot be memorized"[34]. This means that, for instance, jointly querying current data with historical data changing over time is not directly supported and requires the overhead of interfacing with other data management systems, with performance and usability problems. The large number of available commercial DSMSs, such as SQLStream Blaze [32], IBM Infosphere Streams [44] and Oracle Fusion [45] are conceived under the same assumptions.

8.2.1. Joint management of streaming and relational data

With the advent of the Big Data era, a large number of novel data management proposals has tried to revolutionize data management, offering extensible architectures able to cope with different kinds of data, including streams and tabular data. Following the guidelines of the Lambda architecture [46], distributed realtime computation systems such as Apache Storm [47] have appeared and been coupled with distributed computing platforms such as Apache Hadoop

[48] and NoSQL databases such as Amazon Dynamo [49] in order to provide a complete solution to real-time processing and batch processing, but with big disadvantages for users: duplicative development effort in systems offering different data/querying models and languages, and additional overhead for reprocessing/merging data between them. Further systems have then been presented, trying to provide larger flexibility in a single data management solution, as also to the guidelines of the Kappa architecture [50]: big data processing engines such as Apache Spark [3], big data stores such as Apache HBase [4], stream processing frameworks such as Apache Flink [51, 5] and Apache Samza [6]. The extensibility and modularity of their architecture allows them to offer data models often supporting different kinds of data: for instance, Spark data model based on finite Resilient Distributed Datasets (RDDs) is extended towards streaming (infinite) data by means of the SparkStreaming [52] extension. The same applies to their querying model: users can typically implement the kind of operators they want, and, besides typical windowing operators applying to streams, some systems offer the possibility to extend their data and querying model towards the relational world, including for instance aggregates and joins. This is the case, for instance, of StormSQL, SparkSQL [53], SamzaSQL [54], Apache Calcite [55] for Flink, Apache Phoenix [56] for HBase.

On the one hand, such variety of extensions brings great flexibility as to query answering. On the other hand, it essentially introduces fragmentation. From a practical point of view, although offering high performances, such extensions do not provide a real synergic approach to querying streaming and relational data under a unified perspective. For instance, HBase extended with Phoenix supports SQL queries but still not streaming data; SparkStreaming and SparkSQL extensions work together only by introducing substantial overheads (stream data each time needs to be converted and registered as a temporary table to be queried using SQL); similarly, in SamzaSQL and Flink relational operators need to transform the data between the base and relational model, whereas joining a table and a stream requires reading the whole table data in memory ("bootstrap") in order to transform it to a stream. This confirms that (a) the typical drawbacks of DSMSs when dealing with tabular data are still present and (b) the streaming/continuous model and the relational model are kept as two separate worlds.

With reference to the sample domain considered in this paper, also the work in [57] acknowledges the need by financial services of integrating real-time and historical analytics seamlessly on streaming and relational data. [57] presents the implementation of a framework that relies on an extended relational algebra for internal query representation as a means for translating queries expressed in the Q language (a highly domain-specific query language tailored to time series analytics) to SQL. The focus of the paper is on the efficiency of query translation, rather than on providing a comprehensive syntax and semantics of the algebra, and query translation is shown through examples.

Ultimately, these systems certainly have strong scalability and performances in specific use-cases but, since they do not rely on a traditional RDBMS architecture, they are not able to exploit the full expressive power and the well-established

optimizations of RDBMSs' algebra operators.

8.2.2. Systems dealing with time

With regard to temporal capabilities, DSMSs (e.g., [11]) include a timestamp in each data tuple, representing the time the tuple is created. Some stream processing frameworks (e.g., Storm) give the chance either to work without explicit timestamps (i.e., window calculations are performed based on the time when the tuple is processed), or to use source-generated timestamps (e.g., event time, for processing events based on the time when an event occurs). Flink [5] also offers a third option of working with ingestion time (the same as transaction time, i.e., when the data enters the dataflow). However, even if users can typically extend these models (and define custom operators that have to be programmed from scratch), none of the approaches provide the flexibility of automatically maintaining both the streaming data original timestamps and the window evaluation time, as we do when evaluating windowing expressions. By envisioning tuples' original timestamps as streaming data transaction time values, this feature permits the translation of CQs in OTQs and it enables the implementation of the continuous temporal model on a temporal relational engine supporting an extended sequenced semantics.

Another fundamental aspect is time versioning support: while this is typically of little use in a pure DSMS, it instead becomes fundamental when supporting queries involving streams and relational data that changes over time (e.g., in temporal joins). While some systems (e.g., Flink [5] and Samza [54]) underline the importance of the so-called *replay principle* (i.e., a streaming query should produce the same result as the corresponding non-streaming query would if given the same data in a table), none is able to fully support it since their models do not include temporal database semantics and versioning. Many of the above discussed approaches combine streaming features with desirable (T)DBMSs features, however without directly exploiting well-established (T)DBMSs' research and implementations, often with uneven results.

In this respect, an equally hot research topic has been extending standard DBMSs towards a flexible and efficient management of temporal and versioned data: even if some partial temporal support is becoming widespread also in commercial systems [58], much research is still needed in order to fully support such capabilities. For instance, the seminal work [14] provides several breakthroughs towards this direction, basically focused though on the management of temporal sequenced queries.

Quite surprisingly, considering the temporal nature of a data stream, existing data-intensive systems (i.e., both DSMSs and Big Data frameworks) do not support built-in temporal versioning for data. The only exception is HBase/Phoenix and the major standard RDBMSs, where, however, no streaming support is considered.

8.2.3. Discussion of our approach

In this paper we have laid the semantic foundations of a novel unified framework, by grounding our work on two main pillars: our contribution on

streaming tables [13], bringing together for the first time the streaming and temporal database worlds, and the work in [14], presenting a first proof-of-concept that an interval-based temporal DBMS can be successfully implemented on top of standard RDBMS technology. On the one hand, we have brought under a unifying semantic umbrella streaming data as well as temporal relational data, so that they can be queried in a seamless way through the $\mathcal{CTA}$ we introduced. On the other hand, we have extended the approach in [14] by "wrapping" the temporal algebra $\mathcal{TA}$ with a formal framework that translates $\mathcal{CTA}$ expressions involving windowing operators into $\mathcal{TA}$ expressions. To this end, we also extended the $\mathcal{TA}$ in [14], which supports only sequenced temporal queries, to $\mathcal{TA}^{\star}$, which supports also non-sequenced operations within the execution of a sequenced temporal query through the introduction of non-temporal operators. This opens a wider range of querying possibilities, including answering *real temporal queries* (see Appendix for a detailed discussion) as well as OTQs and CQs over streaming, relational and temporal data. As far as we know, this is the first proposal in this direction. Even recent works on semantic aspects of temporal data management in DBMSs (such as [59]) limit their approach to different assumptions (focus on data with a single time dimension and on pure snapshot-reducible algebras).

The definition of $\mathcal{TA}^{\star}$ is quite different from the approach in [60], where the authors aim at reconciling the sequenced and non-sequenced viewpoints by providing a unifying framework where queries can be evaluated using different temporal semantics. Sequenced and non-sequenced ones are considered as variants of a general temporal semantics, which can also be used to realize additional semantics (e.g., context, periodic and preceding semantics). In such an approach, the execution semantics can be selected with annotations preceding the queries (in ATSQL-style) and, thus, different temporal semantics can be selected for the execution of different queries but all the parts of a single query are evaluated with the same semantics. On the contrary, in our approach, individual queries are evaluated with a hybrid semantics, as the $\mathcal{TA}^{\star}$ algebra allows to merge non-sequenced parts into a sequenced $\mathcal{TA}$ expression in order to extend its query expressiveness. Hence, the coexistence of the different temporal semantics is within single queries rather than between different queries, and sequenced and non-sequenced parts can be combined in the same $\mathcal{TA}^{\star}$ expressions to support a wider range of temporal query types (with the same expressive power of a Temporal Relational Calculus, having explicit access to timestamp values, as shown in Appendix A) but also preserving advantages of a sequenced approach. To the best pf our knowledge, this is the first proposal in this direction.

As far as the extension from to $\mathcal{CTA}$ is concerned, an approach similar to ours is followed in [61], where operators for window specification are kept separate from standard algebraic operators. The introduced operators are tailored to deal with a multiset representation of temporal data in a novel logical model and are implemented on ad-hoc data structures for state maintenance under a time-interval approach. Snapshot reducibility is demonstrated for standard operators. Window operators instead overwrite tuples' original timestamps with window evaluation time and thus they are not snapshot-reducible. This approach does not integrate with the theoretical and practical solutions proposed

for the development of a robust temporal database technology, including [14] 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 $\mathcal{TA}^{\star}$.

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 [30, 35], or the most recent one [62], or the time instant at which the join is executed [63]. Operators in $\mathcal{TA}$ follow a precise and commonly adopted sequenced semantics [14], and a temporal join is always performed between pairs of tuples having the same timestamp, thus being valid at the same time instant. The non-sequenced part of expressive temporal queries (e.g., necessary to interoperate data belonging to different temporal snapshots) is captured by the $\mathcal{TA}^{\star}$ extensions we proposed for $\mathcal{TA}$. Further, as noticed about [61], also in [11] 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 concerning the lineage of streaming data, to be used for further processing. Such "provenance witness" timestamps play a role similar to the lineage sets used in [14] to define change-preserving operators. This feature enables $\mathcal{CTA}$ windowing operators to be translated into $\mathcal{TA}$ and, thus, to be snapshot-reducible on valid time.

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 [64] (e.g., SQLStream Blaze [32] considers forward windows in its query syntax specifications but it does not provide an implementation for them). 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.

9. Concluding remarks and future research directions

The ongoing research and the discussed works on the management and querying of streaming, temporal and standard relational data are certainly promising. However, there is still much work to be done in order to have an actual "all-in-one" answer to all the modern application requirements we discussed in the introduction. Following the first step provided by the streaming table concept we recently introduced in [13], in this paper and in its preliminary version [16], we presented a novel temporal algebra extended with windowing and

aggregation operators supporting both OTQs and CQs on streaming, standard and temporal relational data. Continuous queries are proved to be correctly translated into temporal OTQs, thus paving the way to the implementation on a relational engine extended with temporal capabilities. With the aim of avoiding any ambiguity, our approach complies to a formal scheme that guarantees correctness of query rewriting, which is known to have a crucial impact on design and implementation issues like, for instance, query optimization.

Our proposal 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.

In our near 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 or in a traditional DBMS extended with temporal capabilities. Optimization issues include the efficient support of the hybrid sequenced/non-sequenced semantics introduced in this work, requiring techniques for the combined optimization of $\mathcal{TA}^\star$ subexpressions containing non-temporal operators (often involving non-sequenced temporal joins) and $\mathcal{TA}$ expressions which can be dealt with as shown in [14]. A further extension concerns the possibility of seamlessly maintaining two or more times for each tuple. This feature would enable a richer variety of querying possibilities, for instance through operators involving together event and transaction time, as well as out-of-order and skewed inputs' management [22]. Our final aim is to exploit all the consolidated and universally adopted technology a standard temporal framework provides, in order to complete the integration of streaming tables in the context of a temporal RDBMS, thus breaking the traditional barrier between the streaming and the temporal worlds.

A medium-term research direction we intend to investigate concerns the study of metodologies and techniques for modeling/processing linked stream data, a way in which the Linked Data principles can be applied to stream data and be part of the Web of Linked Data [65]. To this end, we will blend our long-standing know-how in the field of semantic data modeling and processing [66, 67], also including temporal aspects [68, 69], with the expertise we have gathered with stream data management [13, 70, 71]. We consider this topic a very promising one that would certainly boost a new range of real-time applications.

A further related topic we are interested in regards reasoning on semantic data streams. Our purpose is to take advantage of the amount of existing work on stream reasoning [72] to study the application of inference techniques to highly dynamic *semantic* data. We deem this issue as a prominent and challenging goal for the definition of innovative means for data analysis in complex environments like, for instance, smart factories in the forthcoming Industry 4.0 scenario.

Another scenario we would like to explore is the management of data streams with intrinsic interval rather than point timestamping. Intelligent sensors can

be thought to produce streams of data concerning the accomplishment of some kind of complete tasks and for which an interval timestamping is needed, as they represent *telic* data [73]. Telic facts (like "the patient had one 100mg intravenous drug infusion from 10:00 to 10:45") are true on an interval but false on any superinterval or subinterval of it, and need to be manipulated with an interval temporal logic or algebra. Extending our framework to the management of interval-based data streams and telic temporal data [74] is another challenge we plan to accept.

In the long term, we also wish to explore the potentialities offered by the availability, at the data model and at the query language level, of complex ¬1NF data types including sets of streaming tables, streaming tables of streaming tables and sets of streaming tables of streaming tables. Such a possibility has been explicitly forbidden in the current approach as it was unnecessary for the purposes of the present work, but could reveal elements of interestingness for particular applications. In this vein, another constraint present in the current approach that could be relaxed concerns the use of sliding window expressions, which could be allowed to also have generic temporal expressions as arguments and to appear inside non-temporal algebra subexpressions.

References

[1] J. Ramnarayan, K. Bachhav, B. Mozafari, S. Wale, S. Menon, N. Kumar, H. Bhanawat, S. Chakraborty, Y. Mahajan, R. Mishra, Snappydata: A hybrid transactional analytical store built on spark, 2016, pp. 2153–2156. doi:10.1145/2882903.2899408.

[2] D. Stripelis, J. L. Ambite, Y.-Y. Chiang, S. Eckel, R. Habre, A scalable data integration and analysis architecture for sensor data of pediatric asthma, Vol. 2017, 2017, pp. 1407–1408. doi:10.1109/ICDE.2017.198.

[3] Apache Spark, `http://spark.apache.org`.

[4] Apache HBase, `http://hbase.apache.org`.

[5] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, K. Tzoumas, Apache flink^{TM}: Stream and batch processing in a single engine, IEEE Data Eng. Bull. 38 (4) (2015) 28–38.

[6] Apache Samza, `http://samza.apache.org`.

[7] U. Cetintemel, J. Du, T. Kraska, S. Madden, D. Maier, J. Meehan, A. Pavlo, M. Stonebraker, E. Sutherland, N. Tatbul, K. Tufte, H. Wang, S. Zdonik, S-Store: A Streaming NewSQL System for Big Velocity Applications, Proc. VLDB 7 (13) (2014) 1633–1636.

[8] Q. Chen, M. Hsu, Cut-and-rewind: Extending query engine for continuous stream analytics, in: A. Hameurlain, J. Kueng, R. Wagner, A. Cuzzocrea, U. Dayal (Eds.), TLDKS XXI, Vol. 9260 of LNCS, Springer, 2015, pp. 94–114.

[9] E. Liarou, S. Idreos, S. Manegold, M. Kersten, Enhanced stream processing in a dbms kernel, in: Proc. of EDBT, 2013, pp. 501–512.

[10] R. Snodgrass (Ed.), The TSQL2 Temporal Query Language, Kluwer, 1995.

[11] A. Arasu, S. Babu, J. Widom, The CQL continuous query language: semantic foundations and query execution, VLDB J. 15 (2) (2006) 121–142.

[12] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases., Addison-Wesley, 1995.

[13] L. Carafoli, F. Mandreoli, R. Martoglia, W. Penzo, Streaming Tables: Native Support to Streaming Data in DBMSs, IEEE Trans. on Systems, Man, and Cybernetics: Systems 47 (10) (2017) 2768–2782.

[14] A. Dignös, M. H. Böhlen, J. Gamper, C. S. Jensen, Extending the kernel of a relational dbms with comprehensive support for sequenced temporal queries, ACM Trans. Database Syst. 41 (4) (2016) 26:1–26:46.

[15] M. H. Böhlen, C. S. Jensen, Sequenced semantics, in: L. Liu, M. T. Özsu (Eds.), Encyclopedia of Database Systems, Second Edition, Springer, 2018. doi:10.1007/978-1-4614-8265-9_1053.
URL https://doi.org/10.1007/978-1-4614-8265-9_1053

[16] F. Grandi, F. Mandreoli, R. Martoglia, W. Penzo, A Relational Algebra for Streaming Tables Living in a Temporal Database World, in: Proc. of TIME Symposium, LIPICS, Mons, Belgium, 2017, pp. 15:1–15:17.

[17] F. Grandi, Temporal interoperability in Multi+Temporal databases, J. Database Manag. 9 (1) (1998) 14–23.

[18] C. S. Jensen, C. E. Dyreson, M. H. Böhlen, J. Clifford, R. Elmasri, S. K. Gadia, F. Grandi, P. J. Hayes, S. Jajodia, W. Käfer, N. Kline, N. A. Lorentzos, Y. G. Mitsopoulos, A. Montanari, D. A. Nonen, E. Peressi, B. Pernici, J. F. Roddick, N. L. Sarda, M. R. Scalas, A. Segev, R. T. Snodgrass, M. D. Soo, A. U. Tansel, P. Tiberio, G. Wiederhold, The consensus glossary of temporal database concepts - february 1998 version, in: S. S. O. Etzion, S Jajodia (Ed.), Temporal Databases: Research and Practice, Springer-Verlag, 1998, pp. 367–405.

[19] J. Chomicki, Temporal query languages: A survey, in: Proceedings of the First International Conference on Temporal Logic, Springer-Verlag, 1994, pp. 506–534.

[20] C. D. Castro, F. Grandi, M. R. Scalas, Semantic interoperability of multitemporal relational databases, in: Proceedings of the 12th International Conference on the Entity-Relationship Approach, Springer-Verlag, 1993, pp. 463–474.

[21] J. Clifford, D. S. Warren, Formal semantics for time in databases, ACM Trans. Database Syst. 8 (2) (1983) 214–254.

[22] U. Srivastava, J. Widom, Flexible Time Management in Data Stream Systems, in: Proc. of ACM PODS, 2004, pp. 263–274.

[23] Insider Trading, `https://www.sec.gov/fast-answers/answersinsiderhtm.html`.

[24] S. Donoho, Early detection of insider trading in option markets, in: Proc. of the ACM KDD, ACM, 2004, pp. 420–429.

[25] M. H. Böhlen, C. S. Jensen, R. T. Snodgrass, Temporal statement modifiers, ACM Trans. Database Syst. 25 (4) (2000) 407–456.

[26] J. Chomicki, D. Toman, M. H. Böhlen, Querying ATSQL databases with temporal logic, ACM Trans. Database Syst. 26 (2) (2001) 145–178.

[27] S. Abiteboul, L. Herr, J. V. der Busche, Temporal versus first-order logic to query temporal databases, in: Proc. of ACM PODS, ACM Press, 1996, pp. 49–57.

[28] D. Toman, D. Niwinski, First-order queries over temporal databases inexpressible in temporal logic, in: Proc. of EDBT, Springer, 1996, pp. 307–324.

[29] A. U. Tansel, Temporal algebras, in: Encyclopedia of Database Systems, Springer US, 2009, pp. 2929–2932.

[30] K. Patroumpas, T. Sellis, Window specification over data streams, in: Current Trends in Database Technology - EDBT 2006 Workshops, 2006, pp. 445–464.

[31] L. Golab, M. T. Özsu, Update-Pattern-Aware Modeling and Processing of Continuous Queries, in: Proc. of ACM SIGMOD, 2005, pp. 658–669.

[32] SQLStream Blaze, `http://sqlstream.com`.

[33] D. Toman, Point vs. interval-based query languages for temporal databases, in: Proc. of ACM PODS, ACM Press, 1996, pp. 58–67.

[34] Y. Bai, H. Thakkar, H. Wang, C. Luo, C. Zaniolo, A data stream language and system designed for power and extensibility, in: Proceedings of the 15th ACM International Conference on Information and Knowledge Management, Proc. of ACM CIKM '06, ACM, 2006, pp. 337–346.

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

[36] S. Chandrasekaran et al., TelegraphCQ: Continuous Dataflow Processing for an Uncertain World, in: Proc. of CIDR, 2003.

[37] J. Chen, D. DeWitt, F. Tian, Y. Wang, NiagaraCQ: A Scalable Continuous Query System for Internet Databases, in: Proc. of ACM SIGMOD, 2000, pp. 379–390.

[38] C. Cranor, T. Johnson, O. Spataschek, V. Shkapenyuk, Gigascope: A stream database for network applications, in: Proc. of ACM SIGMOD, ACM, 2003, pp. 647–651.

[39] E. Panigati, F. A. Schreiber, C. Zaniolo, Data streams and data stream management systems and languages, in: F. Colace, M. De Santo, V. Moscato, A. Picariello, F. A. Schreiber, L. Tanca (Eds.), Data Management in Pervasive Systems, Springer, 2015, pp. 93–111.

[40] D. Toman, On construction of holistic synopses under the duplicate semantics of streaming queries, Vol. 174, 2007, pp. 150–162.

[41] T. Ghanem, A. Elmagarmid, P. Larson, W. Aref, Supporting Views in Data Stream Management Systems, ACM Trans. Database Syst. 35 (1) (2010) 1.

[42] M. J. Sax, G. Wang, M. Weidlich, J.-C. Freytag, Streams and tables: Two sides of the same coin, in: Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics, BIRTE '18, 2018, pp. 1:1–1:10.

[43] N. Laptev, B. Mozafari, H. Mousavi, H. Thakkar, H. Wang, K. Zeng, C. Zaniolo, Extending relational query languages for data streams, in: M. Garofalakis, J. Gehrke, R. Rastogi (Eds.), Data Stream Management: Processing High-Speed Data Streams, Springer, 2016, pp. 361–386.

[44] IBM Infosphere Streams, `http://www.ibm.com/software/products/it/ibm-streams`.

[45] Oracle Fusion, `http://www.oracle.com/it/middleware`.

[46] N. Marz, J. Warren, Big Data. Principles and best practices of scalable realtime data systems, Manning Publications, 2015.

[47] Apache Storm, `http://storm.apache.org`.

[48] Apache Hadoop, `http://hadoop.apache.org`.

[49] Amazon Dynamo, `http://aws.amazon.com/dynamodb`.

[50] Kappa architecture, `http://www.oreilly.com/ideas/questioning-the-lambda-architecture`.

[51] Apache Flink, `http://flink.apache.org`.

[52] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, I. Stoica, Discretized streams: Fault-tolerant streaming computation at scale, in: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, ACM, 2013, pp. 423–438.

[53] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, M. Zaharia, Spark sql: Relational data processing in spark, in: Proc. of ACM SIGMOD, ACM, 2015, pp. 1383–1394.

[54] Y. P. Milinda Pathirage, Julian Hyde, B. Plale, SamzaSQL: Scalable fast data management with streaming SQL, in: Proc. of IEEE IPDP Symposium Workshops, IEEE Computer Society, 2016, pp. 1627–1636.

[55] Apache Calcite, `http://calcite.apache.org`.

[56] Apache Phoenix, `http://phoenix.apache.org`.

[57] L. Antova, R. Baldwin, D. Bryant, T. Cao, M. Duller, J. Eshleman, Z. Gu, E. Shen, M. A. Soliman, F. M. Waas, Datometry hyper-q: Bridging the gap between real-time and historical analytics, in: Proc. of the 2016 Int. Conf. on Management of Data, 2016, pp. 1405–1416.

[58] D. Petkovic, Temporal data in relational database systems: A comparison, in: Á. Rocha, A. M. Correia, H. Adeli, L. P. Reis, M. Mendonça Teixeira (Eds.), New Advances in Information Systems and Technologies, Springer, 2016, pp. 13–23.

[59] A. Dignös, B. Glavic, X. Niu, M. H. Böhlen, J. Gamper, Snapshot Semantics for Temporal Multiset Relations, Proceedings of the VLDB Endowment 12 (6) (2019) 639–652.

[60] C. E. Dyreson, V. A. Rani, A. Shatnawi, Unifying sequenced and non-sequenced semantics, in: Proc. of TIME Symposium, IEEE Computer Society, Kassel, Germany, 2015, pp. 38–46.

[61] J. Krämer, B. Seeger, Semantics and Implementation of Continuous Sliding Window Queries over Data Streams, ACM Trans. Database Syst. 34 (1) (2009) 4.

[62] A. Ayad, J. Naughton, Static Optimization of Conjunctive Queries with Sliding Windows over Infinite Streams, in: Proc of ACM SIGMOD, 2004, pp. 419–430.

[63] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and Issues in Data Stream Systems, in: Proc. of ACM PODS, 2002, pp. 1–16.

[64] F. Zemke, What's new in SQL:2011, SIGMOD Rec. 41 (1) (2012) 67–73.

[65] J. Sequeda, O. Corcho, Linked stream data: A position paper, in: Proc. of the 2nd Int. Conf. on Semantic Sensor Networks, Vol. 522 of SSN'09, CEUR-WS.org, 2009, pp. 148–157.

[66] F. Mandreoli, R. Martoglia, W. Penzo, S. Sassatelli, Data-Sharing P2P Networks with Semantic Approximation Capabilities, IEEE Internet Computing 13 (5) (2009) 60–70.

[67] F. Mandreoli, R. Martoglia, W. Penzo, Approximating expressive queries on graph-modeled data: The GeX approach, Journal of Systems and Software 109 (Supplement C) (2015) 106 – 123.

[68] F. Grandi, Multi-temporal RDF ontology versioning, in: Proc. of Intl' Workshop on Ontology Dynamics (IWOD, in conj. with ISWC), CEUR-WS, Chantilly, VA, 2009.

[69] F. Grandi, T-SPARQL: a TSQL2-like temporal query language for RDF, in: Proc. of Intl' Workshop on on Querying Graph Structured Data (GrapQ10, in conj. with ADBIS), CEUR-WS, Novi Sad, Serbia, 2010, pp. 21–30.

[70] L. Carafoli, F. Mandreoli, R. Martoglia, W. Penzo, A Framework for ITS Data Management in a Smart City Scenario, in: Proc. of the 2nd Int. Conf. on Smart Grids and Green IT Systems SMARTGREENS, 2013, pp. 215–221.

[71] L. Carafoli, F. Mandreoli, R. Martoglia, W. Penzo, A Data Management Middleware for ITS Services in Smart Cities, Journal of Universal Computer Science 22 (2) (2016) 228 – 246.

[72] D. Dell'Aglio, E. D. Valle, F. van Harmelen, A. Bernstein, Stream reasoning: A survey and outlook, Data Science 1 (1-2) (2017) 59–83.

[73] P. Terenziani, R. T. Snodgrass, Reconciling point-based and interval-based semantics in temporal relational databases: A treatment of the telic/atelic distinction, IEEE Trans. Knowl. Data Eng. 16 (5) (2004) 540–551.

[74] P. Terenziani, The impact of the telic\atelic dichotomy on temporal databases, in: Proc. of TIME Symposium, IEEE Computer Society, Kongens Lyngby, Denmark, 2016, p. 3.

Appendix A. Extending the $\mathcal{TA}$ Temporal Algebra to the Support of Temporal Queries

The work [14] has made a breakthrough contribution to the development of the temporal database technology, by showing how an interval-based temporal database can be implemented on top of an existing traditional relational database, by enabling its query engine (with all its optimization machinery) to execute sequenced queries by means of simple kernel extensions.

However, the execution of sequenced queries is only a part of the duties required to a temporal database. In fact, the provision of a query language with a strictly snapshot-reducible semantics [25] is almost useless for the development of temporal applications as it lacks support of real temporal queries. Let us assume for simplicity a time granularity of one day and consider a temporal algebra expression Q^T built with the $\sigma^T, \pi^T, \times^T, \cup^T, -^T$ base operators only. Let Q be a (non-temporal) relational algebra expression obtained from Q^T by replacing each temporal operator with its non-temporal counterpart. Being a temporal database a timestamped collection of snapshot databases, the results

of the execution of Q^T is, by definition, the timestamped collection of the results that can be obtained, day by day, by executing Q on the timeslice of the relations in the database valid at that day (i.e., the results only depend on the database snapshot valid at that day). Hence, there is no need to have a temporal database and a temporal query language to extract the data to build such a result: it is sufficient to have a non-temporal database and execute the same non-temporal query Q every day on the current database instance. A temporal relation is only needed to collect in a persistent way the query results obtained day by day, timestamping them with the current date. On the contrary, a real temporal database with a real temporal query language is needed to execute queries that involve, at the same time, data belonging to different snapshots. We call *real* temporal queries such queries, which are typical non-sequenced queries. For instance, a query retrieving names of employees whose salary has been ever decreased and when, is a typical real temporal query, that needs a real temporal database to be answered. In fact, the answer needs the comparison of the employee salaries belonging to two consecutive snapshots to be determined. A snapshot-reducible temporal algebra is completely useless to this purpose, as it is unable to mix data belonging to different snapshots.

Notice that the necessity of operators with a non-sequenced semantics to express real temporal queries has been rather underestimated in the recent literature regarding temporal algebras. For example, in the the concise survey [29], temporal algebras are classified as based on snapshot evaluation (viz., sequenced semantics) or on traditional evaluation (viz., non-sequenced semantics). Then, in the exemplification of algebras with a snapshot evaluation, the selection operator is defined as $\sigma^t_F(R) = \sigma_F(R_t)$ for all $t \in \mathcal{T}$, where R_t is the snapshot of R at time t and the selection formula F is said to include traditional predicates as well as temporal predicates like *Before, After, Overlaps*, etcetera. Allowing for temporal predicates in F seems clearly aimed at supporting real temporal queries, which is though incompatible with the sequenced semantics: evaluation of temporal predicated in F by $\sigma_F(R_t)$ is impossible as only one snapshot valid at t is available! Hence, only temporal algebras based on traditional or hybrid evaluation are able to express real temporal queries.

The authors of [14] partially acknowledged the limitations of a purely snapshot-reducible algebra and introduced, to overcome them, an extend operator ε and a careful definition of the temporal aggregate operator ϑ^T. The extend operator does *timestamp propagation*, that is it copies the values of the tuple timestamp into an explicit attribute in order to make the timestamp values accessible to the manipulation by other operators. The temporal aggregate operator is a snapshot-reducible operator that is able to produce a result by aggregating data belonging to more than one snapshot (i.e., to all the snapshots spanned by interval timestamps). In particular, they showed in [14] how the use of the extend operator is needed in order to correctly compute the values of temporal aggregates from tuples adopting interval-based timestamping by *scaling* attribute values (in a way depending on their semantics). Attribute scaling is made necessary by the adjustment of intervals which is used to execute snapshot-reducible temporal queries on a traditional database kernel.

However, the support of real temporal queries in [14] is just limited to the evaluation of such temporal aggregates (and the use of the extend operator is basically aimed at the correct computation of such aggregates via the scaling mechanism). For instance, our example query retrieving employees whose salary has been ever decreased cannot be expressed in the temporal algebra proposed in [14]. Let us consider data as stored in an `EMPLOYEE(NAME,SALARY` $\mid T$`)` relation, where T is an interval timestamp as in [14]. In fact, in order to express such a query, pairs of tuples with consecutive interval timestamps have to be found and the salary values in them compared (i.e., a non-sequenced temporal self-join has to be actually executed). Assuming E_1 and E_2 are two aliases for the `EMPLOYEE` relation, the desired result can be obtained by means of the following expression written in the plain relational algebra (accessing timestamps as if they were explicit attributes):

$$\pi_{E_1.\texttt{NAME},E_2.\texttt{START}}(\sigma_{E_1.\texttt{NAME}=E_2.\texttt{NAME}\wedge E_1.\texttt{SALARY}>E_2.\texttt{SALARY}\wedge E_1.T\ \texttt{MEETS}\ E_2.T}(E_1 \times E_2))$$

where $E_1.T\ \texttt{MEETS}\ E_2.T$ means $\texttt{END}(E_1.T) = \texttt{START}(E_2.T)$ [10, Ch. 13].

Using the temporal algebra of [14], the timestamps of E_1 and E_2 can even be propagated via the extend operator to become available as explicit attributes as follows: $F_1 = \varepsilon_U(E_1)$ and $F_2 = \varepsilon_U(E_2)$. But there is no way to test the temporal join condition $F_1.U\ \texttt{MEETS}\ F_2.U$ with the snapshot-reducible operators of the temporal algebra. For instance, the expression:

$$\sigma^T_{F_1.\texttt{NAME}=F_2.\texttt{NAME}\wedge F_1.\texttt{SALARY}>F_2.\texttt{SALARY}\wedge F_1.U\ \texttt{MEETS}\ F_2.U}(F_1 \times^T F_2)$$

always produces an empty result, owing to the snapshot-reducible semantics of the temporal Cartesian product operator, which only combines tuples having an overlapping timestamp (if $F_1.T$ and $F_2.T$ overlap, also their copies $F_1.U$ and $F_2.U$ overlap and, thus, $F_1.U\ \texttt{MEETS}\ F_2.U$ is always false). A snapshot-reducible temporal algebra, by definition, can only interoperate data belonging to the same snapshots, whereas our sample query require data belonging to different snapshots to be interoperated.

This deficiency of all the snapshot-reducible temporal algebras, including that of [14], can be overcome in two ways:

1. by extending the temporal algebra with a timestamp-replacing operator;
2. by extending the temporal algebra with non-temporal algebra operators.

The first solution corresponds to the introduction of a new operator defined as follows:

$$\eta_I(R) := \{(r, \tau') \mid (r, \tau) \in R \wedge \tau' = I(\tau)\}$$

The η operator replaces the timestamps in the tuples of the argument relation R with a time-interval expression I, which can be a constant interval or an interval expression depending on the timestamps of R. For example, $\eta_{[0,\infty)}(R)$ replaces all the timestamps in R with the interval $[0, \infty)$ spanning the whole time domain (with the meaning of valid always), whereas $\eta_{\texttt{SHIFT}(T,1)}(R)$ shifts forward by one time point the timestamps of R (i.e., it replaces $[t_1, t_2)$ with $[t_1 + 1, t_2 + 1)$).

With this new operator, our query retrieving the employees whose salary has been ever decreased can be expressed, for instance, as follows:

$$\pi^T_{F_1.\texttt{NAME},\texttt{START}(F_2.U)}(\sigma^T_{F_1.\texttt{NAME}=F_2.\texttt{NAME}\wedge F_1.\texttt{SALARY}>F_2.\texttt{SALARY}\wedge F_1.U\ \texttt{MEETS}\ F_2.U}(\eta_{[0,\infty)}(F_1)\times^T F_2))$$

In such a case, in practice, the temporal Cartesian product $\times^T$ computes the (non-temporal) Cartesian product between the non-temporal part of F_1 and F_2 and result tuples are timestamped with the timestamp of F_2 (being $[0,\infty) \cap F_2.T = F_2.T$). Hence, the selection condition $F_1.\texttt{NAME} = F_2.\texttt{NAME} \wedge F_1.\texttt{SALARY} > F_2.\texttt{SALARY} \wedge F_1.U\ \texttt{MEETS}\ F_2.U$ can be safely applied and the result projected on $F_1.\texttt{NAME}, \texttt{START}(F_2.U)$ to obtain the names of employees whose salary has even been decreased with the dates it was decreased. The same result can be also obtained as follows:

$$\pi^T_{F_1.\texttt{NAME},\texttt{START}(F_2.U)}(\sigma^T_{F_1.\texttt{NAME}=F_2.\texttt{NAME}\wedge F_1.\texttt{SALARY}>F_2.\texttt{SALARY}\wedge F_1.U\ \texttt{MEETS}\ F_2.U}(\eta_{\texttt{SHIFT}(T,1)}(F_1)\times^T F_2))$$

In such a case, the tuples of F_1 and F_2 with consecutive interval timestamps $[t_1, t_2)$ and $[t_2, t_3)$ will have the timepoint t_2 in common after the shift of the F_1 timestamps effected by η. Thus, they contribute to the Cartesian product results (as $[t_1 + 1, t_2 + 1) \cap [t_2, t_3) = [t_2, t_2 + 1)$), producing a tuple with timestamp $[\texttt{START}(F_2.T), \texttt{START}(F_2.T)+1)$. Also in this case, the required selection condition can be safely applied and the final projection produces the desired results. Notice that the two expressions, which are both correct temporal relations, return the same data but with different timestamps, which are $F_2.T$ in the former case and $[\texttt{START}(F_2.T), \texttt{START}(F_2.T) + 1)$ in the latter case.

The second solution is based on allowing the use of non-temporal algebra operators in temporal algebra expressions. The only requirement for their use, in order to guarantee the closure of the resulting extended temporal algebra, is that they are used, possibly in combination, to produce macro expressions that produce anyway correct temporal relations (whereas their subexpressions may do not). For instance, being F_1 and F_2 the temporal algebra expressions defined above, $\sigma_{\texttt{SALARY}>2000}(F_1)$ and $\pi_{\texttt{NAME}}(F_1) \times F_2$ are correct uses of the non-temporal operators, since their result are temporal relations, whereas $\pi_{\texttt{NAME}}(F_1)$ and $F_1 \times F_2$ are not (the former expression produces a relation without a timestamp and the latter would produce a relation with two timestamps). Hence, the expression $\pi_{\texttt{NAME},\texttt{SALARY},U}(F_1) \times F_2$ represents a correct use of non-temporal operators, as the result is a correct temporal relation (basically the Cartesian product between a relation containing the non-temporal part of the tuples of F_1 and the relation F_2, timestamped with the timestamps of F_2). In particular, such an expression can be used to express our example query as follows:

$$\pi^T_{F_1.\texttt{NAME},\texttt{START}(F_2.U)}(\sigma^T_{F_1.\texttt{NAME}=F_2.\texttt{NAME}\wedge F_1.\texttt{SALARY}>F_2.\texttt{SALARY}\wedge F_1.U\ \texttt{MEETS}\ F_2.U}(\pi_{\texttt{NAME},\texttt{SALARY},U}(F_1)\times F_2))$$

Also the example provided in Sec. 6 is based on the use of this pattern to express a real temporal query.

In our approach, we opted for the second solution for the following reason: since the final target of our work is the implementation of our data model and

query language on a traditional relational database, we preferred to rely on the fact that non-temporal operators are readily available as they are implemented in the kernel of any off-the-shelf relational DBMS. Hence, in defining our $\mathcal{TA}^\star$ algebra in Sec. 3, we decided to resort to their use rather than introducing a new temporal operator like η to be added to the temporal algebra.

During the optimization of a $\mathcal{TA}^\star$ expression, minimal subexpressions representing a correct temporal relation but containing non-temporal operators have to be individuated and processed separately (also adopting custom techniques for the optimized execution of non-sequenced temporal joins). Minimality is referred to the fact that they do not have a temporal algebra operator as root of their syntax tree. For instance, in the sample expression E in Sec. 6, the expression in square brackets is a subexpression of this kind. Then, $\mathcal{TA}^\star$ expressions only containing temporal relations, which can be either base temporal tables, or base streaming tables, or the result of the evaluation of a subexpression involving non-temporal operators, can then be processed and optimized as described in [14]. However, design of detailed optimization strategies and generation of execution plans for general $\mathcal{TA}^\star$ expressions will be addressed in future work.

As remarked by Lemma 1, the $\mathcal{TA}^\star$ extended algebra we proposed in this work is strictly more expressive than a shapshot-reducible algebra like $\mathcal{TA}$. In particular, the $\mathcal{TA}^\star$ extended algebra has the same expressive power of a TSQL2-like temporal SQL extension with explicit reference to temporal attributes (but snapshot-reducible semantics for implicit temporal attributes) like TSQL2 [10], for which it simply represents a syntactic variant. In fact, considering the general temporal SQL SPJ query:

`SELECT` $A_1, A_2, \ldots, A_n, R_k$`.T`
`FROM` $R_1, R_2, \ldots, R_m$
`WHERE` C' `AND` C''

where C' is a selection predicate only involving explicit attributes and C'' is a selection predicate also explicitly involving the timestamps of the relations $R_h, \ldots, R_m$ (with $h \geq 1$), this is equivalent to the $\mathcal{TA}^\star$ expression:

$$\pi^T_{A_1,\ldots,A_n}(\sigma^T_{C'}(R_1 \times^T \cdots \times^T R_{h-1} \times^T \sigma^T_{\bar{C}''}(\pi_{X_h,U}(\varepsilon_U(R_h)) \times \cdots \times \varepsilon_U(R_k) \times \cdots \times \pi_{X_m,U}(\varepsilon_U(R_m)))))$$

where X_i is the explicit schema of $R_i(X_i|T)$ and $\bar{C}''$ it the predicate obtained by substituting references to implicit time `T` with references to its explicit copy added by ε_U. The argument of $\sigma^T_{\bar{C}''}$ is the minimal subexpression containing non-temporal operators in this case. The relation R_k (with $1 \leq k \leq m$), appearing without projection in this subexpression, donates its timestamps to the subexpression result. If no reference to the implicit time of one of the relations $R_h, \ldots, R_m$ is made in the `SELECT` clause, the relation to be used without projection within the subexpression can be any one of the $R_h, \ldots, R_m$ relations. The relations $R_1, \ldots, R_{h-1}$ only involved in C' predicates are managed instead with sequenced semantics, by preserving the main advantages of such an approach (first of all, the fact that overlap conditions between their timestamps are implicit in the translation with temporal algebra operators and do not need to be explicited by users at the query language level).

The correspondence between any general $\mathcal{CTA}$ expression and its SQL-like syntax as exemplified in Sec. 6 can be defined in a similar way, also taking into account grouping, window declarations and flattening/aggregate expressions, and sampling specifications.

Appendix B. Expressing the Snapshot-equality Query in the Extended Temporal Algebra $\mathcal{TA}^\star$

The "snapshot-equality" query tests for the existence of distinct database snapshots with an equal content. Assuming the database schema composed of a single unary relation $S(X|T)$ using point timestamping, it can easily be expressed in TRC as follows:

$$(\exists t)(\exists t')[t \neq t' \wedge (\forall x)(S(x,t) \equiv S(x,t'))]$$

Such a query cannot be expressed in FOTL or in a standard temporal algebra acting on relations with a single implicit timestamp [27, 28]. On the contrary, we will show in the following how it can be expressed in our extended temporal algebra $\mathcal{TA}^\star$ also employing non-temporal operators.

The expression $\pi_{X,U_1}(\varepsilon_{U_1}(S))$ represents a non-temporal relation obtained from S by transforming the implicit timestamping attribute T into an explicit attribute U_1. Such a relation is non-temporal as the timestamp column has been projected out. Hence, if we use such a relation as argument of a $\mathcal{TA}$ operator, it is virtually converted to a temporal relation as shown in Sec. 2 (i.e., obtained as the Cartesian product between the tuples of the non-temporal relation and the whole time domain $\mathcal{T}$). Then we can consider the temporal division operator $\div^T$, defined as the temporal extension with sequenced semantics of its non-temporal counterpart $\div$ (the semantics of $\div$ is the standard one considered in relational algebra; for example, the division between $R_1(X_1X_2)$ and $R_2(X_2)$ can be defined as $R_1 \div R_2 = \pi_{X_1}(R_1) - \pi_{X_1}[(\pi_{X_1}(R_1) \times R_2) - R_1])$. In our case, the temporal division is used to express the required universal quantification on values of the explicit attribute X. In fact, we can use the expression:

$$P = \pi_{X,U_1}(\varepsilon_{U_1}(S)) \div^T S$$

whose result is a temporal relation with schema $P(U_1, T)$, to find the pairs (t_i, t_j) of timepoints, such that the snapshot of S valid at t_i contains all the X values present in the snapshot of S valid at t_j.

Notice that, owing to the sequenced semantics, the tuples valid at time t in P are the ones that can be computed by executing the non-temporal division $\div$ between $\pi_{X,U_1}(\varepsilon_{U_1}(S))$ and the explicit contents of the timeslice at time t of S (i.e., $\pi_X(\tau_t(S))$). The results of such a non-temporal division are the values t' of U_1 for which all the values x of X in $\tau_t(S)$ are present as explicit tuples (x, t') in S (actually in the snapshot valid at t of the temporal conversion of $\pi_{X,U_1}(\varepsilon_{U_1}(S))$; notice that, owing to the conversion, the explicit contents of all the snapshots are equal to the non-temporal relation $\pi_{X,U_1}(\varepsilon_{U_1}(S))$).

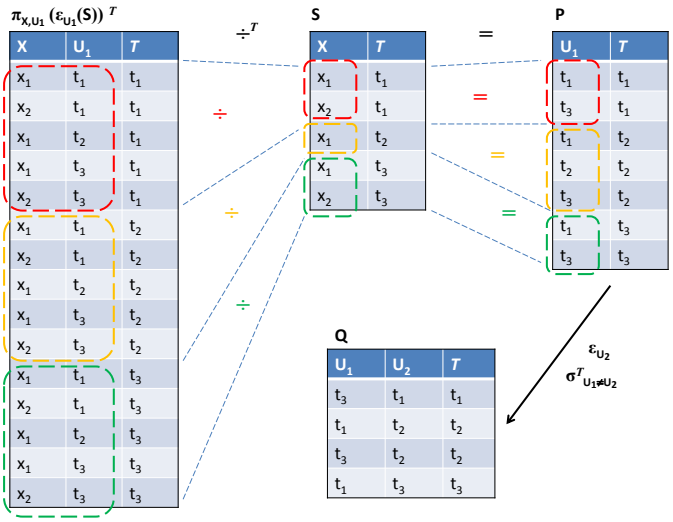


Figure B.4: Example of computation of $\pi_{X,U_1}(\varepsilon_{U_1}(S)) \div^T S$ and further manipulations leading to $Q(U_1, U_2|T)$. The table $\pi_{X,U_1}(\varepsilon_{U_1}(S))^T$ to the left represents the tuples of interest in the result of the temporal conversion of $\pi_{X,U_1}(\varepsilon_{U_1}(S))$ (all the other tuples produced by the conversion, with timestamps different form t_1, t_2 and t_3, do not contribute to the result of the temporal division owing to the sequenced semantics and, thus, have been omitted in the figure). The red, yellow and green colors highlight the $\div$ computations made on the individual snapshots valid at t_1, t_2 and t_3, respectively, to assemble the final $\div^T$ result in accordance to the sequenced semantics.

The timestamp T of P can also be made explicit for further manipulations as follows:

$$Q = \sigma^{T}_{U_1 \neq U_2} \left(\varepsilon_{U_2} [\pi_{X,U_1}(\varepsilon_{U_1}(S)) \div^{T} S] \right)$$

The outer selection eliminates tuples with $U_1 = U_2$, representing trivial snapshot self-containments. An example of the functioning of the temporal division (with further manipulations leading to Q) can be found in Fig. B.4.

In order to find equal snapshots, we must finally test for reciprocal containment of the X values present in the snapshots valid at t_i and t_j, which means to test for the presence of both (U_1, U_2) pairs (t_i, t_j) and (t_j, t_i) in Q. This can be done in more than one way with the operators of the temporal algebra, for instance using attribute rename and intersection or (semi)join. Using rename and intersection (which in $\mathcal{TA}$ can be defined, for example, as $R_1 \cap^{T} R_2 = R_1 -^{T} (R_1 -^{T} R_2)$), the following expression:

$$\pi_{U_1,U_2}(Q) \cap^{T} \rho_{U_1,U_2 \leftarrow U_2,U_1}(Q)$$

where the operator $\rho_{U_1,U_2 \leftarrow U_2,U_1}$ is used to rename the attributes U_1 to U_2 and U_2 to U_1 in Q, respectively, returns a non-empty result if and only if the TRC snapshot-equality query returns a true value. In fact, the (U_1, U_2) pairs in the tuples of the result are the timestamps of the equal snapshots. Notice that $\pi_{U_1,U_2}(Q)$ is a non-temporal relation, which is converted to a temporal relation (with tuples valid always) since it appears as argument of a temporal operator in the expression above. In conclusion, the following Lemma holds:

Lemma 2. *The "snapshot-equality" query can be expressed in the extended temporal algebra $\mathcal{TA}^{\star}$.*

Finally, notice that considering point timestamping in S is not a limitation, as an expression similar to the proposed solution can also be easily found for answering the snapshot-equality query on a relation using interval timestamping.