LNCS 5071, pp 257-269, July 2008.
(URL: http://www.springerlink.com/content/xx4718762600/)
Copyright © Springer-Verlag

Towards General Temporal Aggregation

Michael H. Böhlen[1], Johann Gamper[1], and Christian S. Jensen[2]

[1] Free University of Bozen-Bolzano, Italy
{boehlen,gamper}@inf.unibz.it
[2] Aalborg University, Denmark
csj@cs.aau.dk

Abstract. Most database applications manage time-referenced, or temporal, data. Temporal data management is difficult when using conventional database technology, and many contributions have been made for how to better model, store, and query temporal data. Temporal aggregation illustrates well the problems associated with the management of temporal data. Indeed, temporal aggregation is complex and among the most difficult, and thus interesting, temporal functionality to support. This paper presents a general framework for temporal aggregation that accommodates existing kinds of aggregation, and it identifies open challenges within temporal aggregation.

1 Introduction

In database management, aggregation refers to the process of consolidating, or summarizing, a database instance; this is typically done by creating so-called *aggregation groups* of elements in the database and then applying an aggregation function, e.g., avg, $count$, or min, to each group, thus obtaining an aggregate value for each group.

In early work, Klug [6] put forward a formal relational database framework that encompassed aggregation. In his framework, aggregation is performed according to two parameters: (1) a set of attributes drawn from an argument relation, termed grouping attributes, and (2) pairs of a new attribute name and an aggregation function. The tuples in the relation are partitioned so that tuples with identical values for the grouping attributes are assigned to the same group. For each of the resulting aggregation groups, each aggregation function is evaluated on the tuples in the group, and the result is stored as a value of the associated attribute for each tuple in the group.

In temporal databases, tuples are typically stamped with time intervals that capture the valid time of the information, or facts, they record. During the 1980's, aggregation was incorporated in several query languages, e.g., Ben-Zvi's Time Relational Model [1], Navathe and Ahmed's TSQL [7], Snodgrass' TQuel [8], and a proposal by Tansel [10]. Some of these advances were subsequently consolidated in the TSQL2 proposal [9].

When aggregating temporal relations, it is meaningful to also group the tuples according to their timestamp values. With temporal grouping, groups of values from the time domain are formed. A tuple is then assigned to each group that overlaps with its timestamp, this way obtaining groups of tuples. When an aggregation function is applied to the groups of tuples, a temporal relation results. Different kinds of temporal groupings have emerged as being important. In *instant* temporal aggregation, the time domain is partitioned into time instants, or points. In *moving-window* (or cumulative) temporal aggregation, additionally a time period is placed around each time instant to determine the aggregation groups. With *span* aggregation, the time line is partitioned into user-defined time periods.

This paper presents a general model for temporal aggregation that extends Klug's framework and that subsumes the temporal approaches mentioned above. The model provides orthogonal support for two aspects of aggregation: (a) the definition of *partial result tuples* for which to report one or more aggregate values, and (b) the definition of *aggregation groups*, i.e., the collections of argument tuples that are associated with the partial result tuples and over which the aggregation functions are to be computed. Aggregation then takes three parameters: a partial result relation, $\mathbf{g}$; a mapping function, θ; and a set of pairs of an aggregation function and an attribute name, f_i/C_i.

The most related, past works are due to Vega Lopez et al. [11] and Böhlen et al. [3]. The former offers a framework that enables the analysis and comparison of different forms of temporal aggregation based on various mechanisms for defining aggregation groups, which all take advantage of different granularities. This leads to a point-based view that is not capable to preserve lineage information, and the resulting aggregation groups are contiguous in the time dimension, i.e., the union of the timestamps of all tuples in an aggregation group forms a convex set of time points. The latter offers a framework that decouples the partitioning of the time domain from the specification of the aggregation groups. This paper's proposal builds on this work and extends it in several directions. We elaborate on the relation to Klug's and SQL's framework, show how to express previous forms of temporal aggregation in the general model, and discuss by examples the additional expressiveness of the general model.

We proceed to introduce a running example. Section 3 then defines the new model, and Section 4 illustrates how important kinds of temporal aggregation can be defined using the model. Section 5 proceeds to identify directions for further research in temporal aggregation. Section 6 summarizes the paper.

2 Aggregation Example

Consider a temporal relation $\mathbf{emp}$ that captures work contracts with employees, recording for each contract the name of the employee (N), a contract identifier (CID), the department to which the employee is assigned for the duration of the contract, the monthly salary for the contract period (S), and the valid time of the contract (T). An instance of this relation is shown Fig. 1(a) and illustrated graphically in the upper part of Fig. 2, where the horizontal lines indicate the valid-time intervals of the tuples.

We consider the following three temporal aggregation queries over the relation:

– Q_{ITA}: *For each month and department, what is the number of contracts?*
– Q_{MWTA}: *For each month, how many contracts have been in effect during this month and the preceding two months?*
– Q_{STA}: *For each half-year period and department, what is the number of contracts?*

Q_{ITA} exemplifies instant temporal aggregation, for which the aggregation is applied to each database state, in this case to each month. To compute the aggregate result for

	N	CID	D	S	T
r_1	Jan	140	DB	1200	[1,12]
r_2	Dan	141	DB	700	[1,5]
r_3	Dan	150	DB	700	[6,15]
r_4	Tim	143	AI	2000	[4,9]

(a) Relation **emp**

D	Cnt	T
DB	2	[1,5]
DB	2	[6,12]
DB	1	[13,15]
AI	1	[4,9]

(b) Q_{ITA}

D	Cnt	T
DB	2	[1,5]
DB	3	[6,7]
DB	2	[8,14]
DB	1	[15,17]
AI	1	[4,11]

(c) Q_{MWTA}

D	Cnt	T
DB	3	[1,6]
DB	2	[7,12]
DB	1	[13,18]
AI	1	[1,6]
AI	1	[7,12]

(d) Q_{STA}

Fig. 1. Temporal Relation **emp** and Different Aggregation Queries

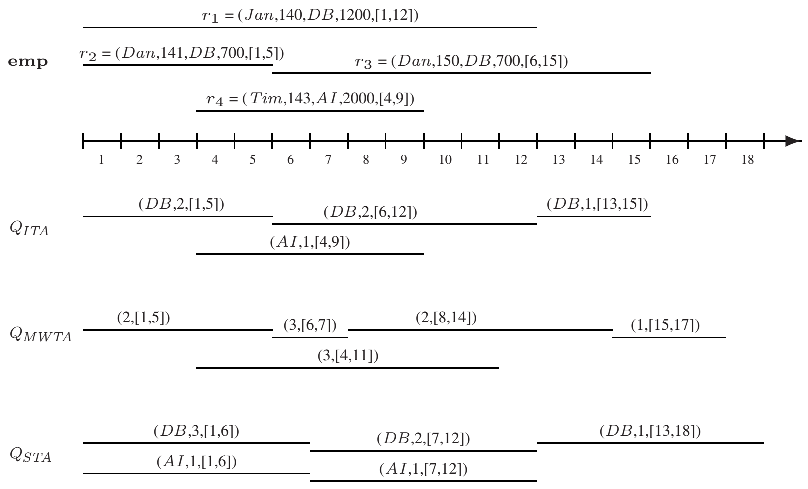

Fig. 2. Graphical Representation of the **emp** Relation and Aggregation Queries

a specific month, all tuples that are valid for that month are considered. Coalescing is used to get an interval-timestamped result relation. Coalescing yields result tuples over maximal time intervals, also called *constant intervals*. For forming maximal intervals two options exist. Either, the coalescing is performed wrt. the aggregate value alone, or it is performed wrt. the aggregate value and the lineage, i.e., the set of argument tuples used for computing the aggregate value. Coalescing with lineage preservation is the most general approach and is thus used here [2]. The result of Q_{ITA} is shown in Fig. 1(b) and graphically illustrated in Fig. 2. Note that without lineage preservation, $(DB, 2, [1, 5])$ and $(DB, 2, [6, 12)$ would have been merged.

Q_{MWTA} illustrates moving-window aggregation. Here, the aggregate value for each month is computed over all tuples that overlap this month or one of the preceding two months. Thus, the last result tuple extends beyond the end point of the last argument tuple. To obtain result tuples over maximal intervals, coalescing is applied similarly to how it is done for ITA. The result of Q_{MWTA} is shown in Fig. 1(c) and graphically illustrated in Fig. 2.

Q_{STA} is a span aggregation query. The time domain is first partitioned into half-year intervals independently of the argument relation. Then, for each half-year interval, the aggregation function is computed over all argument tuples that overlap that half year. The result of Q_{STA} is shown in Fig. 1(d) and graphically illustrated in Fig. 2.

3 General Temporal Aggregation

3.1 Preliminaries

We assume a discrete *time domain*, Δ^T, consisting of a totally ordered set of elements, termed time points (or instants). We assume a data model in which a timestamp, T, is assigned to each tuple that captures when the corresponding fact was, is, or will be true in the modeled reality. A timestamp is a convex set over the time domain and is represented by two time points, $[T_s, T_e]$, denoting its inclusive starting and ending points, respectively. In short, we assume a valid-time data model in which tuples are timestamped with intervals.

A *relation schema* is a three-tuple $R = (\Omega, \Delta, dom)$, where Ω is a non-empty, finite set of attributes, Δ is a finite set of domains, and $dom : \Omega \rightarrow \Delta$ is a function that associates a domain with each attribute. A *temporal relation schema* is a relation schema with at least one timestamp valued attribute, i.e., $\Delta^T \in \Delta$. A *tuple r over schema R* is a function that maps every $A_i \in \Omega$ to a value $v_i \in dom(A_i)$. A *relation* $\mathbf{r}$ *over schema R* is a finite set of tuples over R.

For notational simplicity, we assume an ordering of the attributes and represent a temporal relation schema as $\mathbf{r} = (A_1, \ldots, A_n, T)$ and a corresponding tuple as $r = (v_1, \ldots, v_n, [T_s, T_e])$. For a tuple r and an attribute A_i we write $r.A_i$ to denote the value of the attribute A_i in r. For a set of attributes $A_1, \ldots, A_k$, $k \leq n$, we define $r[A_1, \ldots, A_k] = (r.A_1, \ldots, r.A_k)$.

3.2 A General Model of Temporal Aggregation

Recall that Klug's (and SQL's) conventional framework for non-temporal aggregation performs aggregation on an argument relation according to two parameters [6]:

1. A set of attributes drawn from the argument relation, termed grouping attributes
2. A set of pairs of a new attribute name and an aggregation function

The tuples in the argument relation are partitioned according to their values for the grouping attributes. Then for each partition, each aggregation function given in the second parameter is computed on the tuples in the partition, and the result is stored as a value of the associated attribute for each tuple in the partition. The non-grouping attributes of the argument relation may be eliminated from the result by means of a projection using relational algebra.

The new model for temporal aggregation extends Klug's framework to the temporal context and generalizes it to provide orthogonal support for two important aspects of aggregation: (a) the definition of partial result tuples for which to report one or more aggregate values, and (b) the definition of aggregation groups, i.e., collections of argument tuples that are associated with the result groups and over which the aggregation functions are computed.

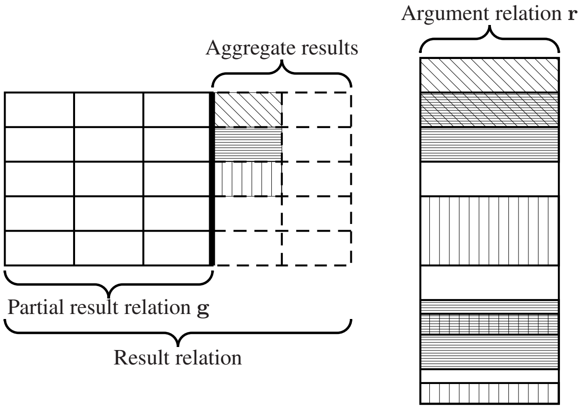
Argument relation $\mathbf{r}$
Aggregate results
Partial result relation $\mathbf{g}$
Result relation

Fig. 3. General Temporal Aggregation

We assume that the aggregation is applied to a relation $\mathbf{r}$, as described earlier. The new temporal aggregation model allows then to specify the following three parameters: (1) a *partial result relation*, $\mathbf{g}$, (2) a *mapping function*, θ, from $\mathbf{r}$ to $\mathbf{g}$, and (3) a set of *aggregation functions*, $\mathbf{F}$. The aggregation model is illustrated in Fig. 3.

Instead of partitioning the tuples in the argument relation according to their values for certain of their attributes, we introduce a separate *partial result relation*, $\mathbf{g}$, that contains a partial result tuple for each tuple that will be included in the result relation; i.e., these tuples will be extended with the aggregate results to form final result tuples. The partial result relation has schema $G = (B_1, \ldots, B_m, T)$, where the B_i are non-temporal attributes and T is a timestamp attribute that specifies a time interval (or time point, as a special case of an interval) over which to report an aggregation result. This relation generally has as attributes a subset of the attributes of the argument relation, the timestamp attribute being one of them. Thus, it can typically be specified as a relational algebra expression over the argument relation, i.e., $\mathbf{g} = RA(\mathbf{r})$. In general, however, the attributes B_i and the timestamp T in the partial result relation may also be obtained from relations other than $\mathbf{r}$.

The second parameter, *mapping function*, $\theta : \mathbf{r} \rightarrow \mathbf{g}$, maps tuples from the argument relation, $\mathbf{r}$, to tuples in the partial result relation, $\mathbf{g}$. It may assign the same argument tuple to zero, one, or many partial result tuples. In other words, function θ associates with each partial result tuple a set of argument tuples, termed its *aggregation group*, over which to compute the aggregates to be reported for that tuple. This differs from the conventional framework, where each input tuple is mapped to exactly one group, based on equal values over all grouping attributes.

The third parameter is retained from the conventional framework and specifies the *aggregation functions*, $\mathbf{F} = \{f_1/C_1, \ldots, f_k/C_k\}$. Each f_i is some aggregation function that takes a (temporal) relation as argument and applies aggregation to one of the relation's attributes. The resulting value is stored as the value of an attribute named C_i. For instance, the pair $count_{CID}/Cnt$ states that $count_{CID}$ counts the CID values in the argument relation and returns the count, which is stored as a value of attribute Cnt. Using this notation, we allow for a family of count functions, one for each attribute of the argument relation. For example, $count_N$ and $count_S$ counts over the name and the salary attribute, respectively.

Definition 1 (General Temporal Aggregation). Let $\mathbf{g}$ be a partial result relation, θ a mapping function, and $\mathbf{F}$ a set of aggregation functions, as introduced earlier. The *general temporal aggregation* is then defined as follows:

$$\mathcal{G}^T[\mathbf{g}, \theta, \mathbf{F}]\mathbf{r} = \{g \circ f \mid g \in \mathbf{g} \wedge \mathbf{r}_g = \{r \in \mathbf{r} \mid \theta(r) = g\} \wedge f = f_1(\mathbf{r}_g), \ldots, f_k(\mathbf{r}_g)\}$$

The schema of the result relation is $(B_1, \ldots, B_m, C_1, \ldots, C_k, T)$.

The mapping function, θ, defines and associates an aggregation group, $\mathbf{r}_g \subseteq \mathbf{r}$, with each partial result tuple, $g \in \mathbf{g}$. The aggregation functions are computed over these aggregation groups. The final result tuples are given as the partial result tuples extended ($\circ$, concatenation) with the results of the aggregation functions.

General temporal aggregation decouples the specification of the desired result tuples (i.e., the partial result tuples) from the specification of the aggregation groups (i.e., the mapping of argument tuples to the partial result tuples). In SQL and Klug's framework, the desired result tuples and the aggregation groups are determined by the grouping attributes only. Each different combination of grouping attribute values forms then a partial result tuple and—by equality on the attribute values—determines a corresponding aggregation group. We believe that the specification of the partial result tuples should be decoupled from the specification of the associated aggregation groups, and we find it natural to allow for the use of other operators than simply equality comparison for the specification of the aggregation groups. This yields a more flexible and expressive framework for temporal aggregation.

An important aspect of the framework is that the values for the timestamp attribute in the partial result tuples may be either fixed and provided by the user, or it may be inferred from the data in the argument relation. The use of *fixed intervals* corresponds to how the non-timestamp attribute values are treated: they must be provided explicitly. The use of *inferred intervals* is unique to the timestamp attribute. An inferred interval in a partial result tuple is calculated as the intersection of the intervals of the argument tuples that contribute to the aggregate results to be associated with that partial result tuple. These inferred intervals are termed *constant* because there are no changes in the argument relation during these intervals. Constant intervals are non-overlapping and maximal. Queries Q_{ITA} and Q_{STA} illustrate the difference between user-provided and inferred intervals.

The new model is quite general. The partial result relation, $\mathbf{g}$, is completely independent of the argument relation, $\mathbf{r}$, and its only purpose is to group the results. This provides extensive flexibility in arranging the results according to various criteria, and it makes it possible to express different forms of temporal aggregates including the ones proposed previously. We will show this next.

4 Different Forms of Temporal Aggregation

In part to explore the use and generality of the proposed aggregation framework, we show how three previously proposed forms of temporal aggregation can be expressed in a uniform manner using the framework. We also discuss aggregation queries that are difficult or even impossible to express in terms of the traditional temporal aggregation operators, but can be expressed easily in the new framework.

4.1 Instant Temporal Aggregation

In *instant temporal aggregation* (ITA), the time domain is partitioned into time instants, and an aggregation group is associated with each time instant t that contains all tuples with a timestamp that contains t. Then the aggregation functions are evaluated on each group, producing each a single aggregate value for each t. Finally, identical aggregate results for consecutive time instants are coalesced into the previously mentioned constant intervals.

In some approaches, the aggregate results for a constant interval must also have the same lineage, meaning that they are produced from the same set of argument tuples. Query Q_{ITA} and its result in Fig. 1(b) illustrate ITA. Without the lineage requirement, the result tuples $(DB, 2, [1, 5])$ and $(DB, 2, [6, 12])$ would become $(DB, 2, [1, 12])$.

Definition 2 (Instant Temporal Aggregation). Let $\mathbf{r}$ be a temporal relation, $\mathbf{F}$ be a set of aggregation functions, and $A = A_1, \ldots, A_k$ be the grouping attributes in $\mathbf{r}$. Further, let $\mathbf{s} = \pi[A, T_s]\mathbf{r} \cup \pi[A, T_e{+}1/T_s]\mathbf{r}$ be the start points and $\mathbf{e} = \pi[A, T_s{-}1/T_e]\mathbf{r} \cup \pi[A, T_e]\mathbf{r}$ be the delimiting points of the constant intervals. Then the *instant temporal aggregation* for the aggregation functions in $\mathbf{F}$ over the argument relation $\mathbf{r}$ grouped by A can be expressed in the general temporal aggregation model as $\mathcal{G}^T[\mathbf{g}, \theta, \mathbf{F}]\mathbf{r}$, where:

$$\begin{aligned} \mathbf{g} &= \pi[A, [T_s, min(T_e)/T_e]](\mathbf{s} \bowtie[\mathbf{s}.A = \mathbf{e}.A \wedge T_s \leq T_e]\, \mathbf{e}) \\ \theta(r) &= \{g \in \mathbf{g} \mid g.A = r.A \wedge g.T \cap r.T \neq \emptyset\} \end{aligned}$$

To express ITA, the partial result relation, $\mathbf{g}$, needs to specify the constant intervals of the result tuples, considering also the grouping attributes, $A_1, \ldots, A_k$. First, $\mathbf{s}$ and $\mathbf{e}$ collect all start and end points of the constant intervals together with the grouping attribute values. Each argument tuple, $r \in \mathbf{r}$, induces two start points (the tuple's start point, $\mathbf{r}.T_s$, and the successor of the tuple's end point, $\mathbf{r}.T_e{+}1$) and two end points (the tuple's end point, $\mathbf{r}.T_e$, and the predecessor of the tuple's start point, $\mathbf{r}.T_s{-}1$). Second, those pairs of start and end points are selected that form a valid constant interval. This is the case if for each start point the closest end point that is greater than or equal to the start point is selected. This can be expressed as a join followed by a generalized projection.

Example 1. Consider Query Q_{ITA}. The start and end points of the constant intervals are given as $\mathbf{s} = \{(DB, 1), (DB, 6), (DB, 13), (DB, 16), (AI, 4), (AI, 10)\}$ and $\mathbf{e} = \{(DB, 0), (DB, 5), (DB, 12), (DB, 15), (AI, 3), (AI, 9)\}$, respectively. Substituting $\mathbf{s}$ and $\mathbf{e}$ in the expression for the partial result relation, we get $\mathbf{g} = \{(DB, [1, 5]), (DB, [6, 12]), (DB, [13, 15]), (AI, [4, 9])\}$. The aggregation functions are $\mathbf{F} = \{count_{CID}/Cnt\}$, and the mapping function is $\theta(r) = \{g \in \mathbf{g} \mid g.D = r.D \wedge g.T \cap \mathbf{emp}.T \neq \emptyset\}$. To compute, for example, the aggregate value over the constant interval $[1, 5]$, the mapping function selects the two argument tuples r_1 and r_2.

This definition of ITA preserves lineage: adjacent result tuples with the same aggregate value are not coalesced if they are derived from different argument tuples (cf. the first two result tuples of Q_{ITA} for the DB department).

4.2 Moving-Window Temporal Aggregation

With *moving-window temporal aggregation* (MWTA) (first introduced in TSQL [7] and later also termed *cumulative temporal aggregation* [8,12]), a time window is used to determine the aggregation groups. For each time instant t, an aggregation group is defined as the set of argument tuples that hold in the interval $[t-w, t]$, where $w \geq 0$ is a window offset. In some work [11], a pair of offsets w and w' is used, yielding a window $[t-w, t+w']$ for determining the aggregation groups. After computing the aggregation functions for each aggregation group, coalescing is applied similarly to how it is done for ITA to obtain result tuples over maximal time intervals.

Query Q_{MWTA} and its result in Fig. 1(c) illustrate MWTA. To answer this query, a window is moved along the time line, computing at each time point an aggregate value over the set of tuples that are valid at some point during the last three months.

While both ITA and MWTA partition the time domain into time instants, they differ in how the aggregation groups for each time instant are defined.

Definition 3 (Moving-Window Temporal Aggregation). Assume the earlier definitions of $\mathbf{r}$, $\mathbf{F}$, and $A = A_1, \ldots, A_k$, and let w be a non-negative window offset. Further, let $\mathbf{s} = \pi[A, T_s]\mathbf{r} \cup \pi[A, T_e + w/T_s]\mathbf{r}$ be the start points and $\mathbf{e} = \pi[A, T_s - 1/T_e]\mathbf{r} \cup \pi[A, T_e + w - 1/T_e]\mathbf{r}$ be the end points of the constant intervals. Then the *moving-window temporal aggregation* for the aggregation functions in $\mathbf{F}$ over relation $\mathbf{r}$ grouped by A and using window offset w can be expressed as $\mathcal{G}^T[\mathbf{g}, \theta, \mathbf{F}]\mathbf{r}$, where:

$$\mathbf{g} = \pi[A, [T_s, min(T_e)/T_e]](\mathbf{s} \bowtie[\mathbf{s}.A = \mathbf{e}.A \wedge T_s \leq T_e]\, \mathbf{e})$$
$$\theta(r) = \{g \in \mathbf{g} \mid g.A = r.A \wedge [g.T_s - w + 1, g.T_e] \cap r.T \neq \emptyset\}$$

The expression of MWTA is similar to that of ITA; the only difference is that the effect of the window offset, w, must be considered both for the computation of the constant intervals that are stored in the partial result relation, $\mathbf{g}$, and in the mapping function, θ. Intuitively, each argument tuple affects the aggregation result beyond its own timestamp. Thus, to determine $\mathbf{s}$ and $\mathbf{e}$ to generate the timestamps of the partial result tuples, the window offset, w, is added to the end points of the argument tuples. The mapping function, θ, is modified similarly; the only difference is that the start point of the partial result tuple is decreased by w in order to collect also argument tuples that do not overlap with the timestamp of the result tuple, but have to be considered for the computation of the aggregates.

Example 2. Consider Query Q_{MWTA}, which has a window offset of 3. The start points of the constant intervals together with the grouping attribute values are $\mathbf{s} = \{(DB, 1), (DB, 6), (DB, 8), (DB, 15), (DB, 18), (AI, 4), (AI, 12)\}$ and the end points $\mathbf{e} = \{(DB, 0), (DB, 5), (DB, 7), (DB, 14), (DB, 17), (AI, 3), (AI, 11)\}$.

Substituting $\mathbf{s}$ and $\mathbf{e}$ in the expression for the partial result relation, we get $\mathbf{g} = \{(DB, [1,5]), (DB, [6,7]), (DB, [8,14]), (DB, [15,17]), (AI, [4,11])\}$. The aggregation functions are $\mathbf{F} = \{count_{CID}/Cnt\}$, and the mapping function is $\theta(r) = \{g \in \mathbf{g} \mid g.D = r.D \wedge [g.T_s - 2, g.T_e] \cap \mathbf{emp}.T \neq \emptyset\}$. To compute, for example, the aggregate value over the constant interval $[6,7]$, the mapping function uses the argument tuples r_1, r_2, and r_3.

4.3 Span Temporal Aggregation

For *span temporal aggregation* (STA), the time domain is first partitioned into predefined intervals that are defined independently of the argument relation. For each such interval, an aggregation group is then given as the set of all argument tuples that overlap the interval. A result tuple is produced for each interval by evaluating an aggregation function over the corresponding aggregation group.

Query $Q_{STA}$and its result in Fig. 1(d) illustrate STA. The pre-defined intervals are 6-month periods.

Unlike in ITA and MWTA, the timestamps of the result tuples in STA are specified independently of the argument data. Most approaches consider only regular time spans expressed in terms of granularities, e.g., years, months, and days.

Definition 4 (Span Temporal Aggregation). Assume the earlier definitions of $\mathbf{r}$, $\mathbf{F}$, and $A = A_1, \dots, A_k$, and let $\mathbf{p}$ be a relation with a single attribute T that contains the time intervals over which to report result tuples. Then *span temporal aggregation* can be expressed as $\mathcal{G}^T[\mathbf{g}, \theta, \mathbf{F}]\mathbf{r}$, where:

$$\mathbf{g} = \pi[A]\mathbf{r} \times \mathbf{p}$$
$$\theta(r) = \{g \in \mathbf{g} \mid g.A = r.A \wedge g.T \cap r.T \neq \emptyset\}$$

In the expression of STA, we assume that the timestamps of the result tuples are given in a relation $\mathbf{p}$. This relation is joined with the argument relation, $\mathbf{r}$, and projected to the grouping attributes, A, and the timestamp attribute, T, to form the partial result relation, $\mathbf{g}$. The mapping function, θ, is the same as for ITA.

Example 3. Consider Query Q_{STA}, which reports a result tuple for each six-month period. The time intervals of the result tuples are then given as $\mathbf{p} = \{([1,6]), ([7,12]), ([13,18])\}$, which gives a partial result relation $\mathbf{g} = \{(DB, [1,6]), (DB, [7,12]), (DB, [13,18]), (AI, [1,6]), (AI, [7,12]), (AI, [13,18])\}$. The aggregation functions are $\mathbf{F} = \{count_{CID}/Cnt\}$, and the mapping function is $\theta(r) = \{g \in \mathbf{g} \mid g.D = r.D \wedge g.T \cap \mathbf{emp}.T \neq \emptyset\}$. To compute, e.g., the aggregate value over the period $[1,6]$, the mapping function uses the tuples r_1, r_2, and r_3.

Note that STA reports a result tuple for all predefined intervals. If the aggregate group is empty, the aggregate value is 0 or NULL. This behavior can be controlled by adjusting the definition of $\mathbf{g}$.

4.4 Aggregation over Non-contiguous Aggregation Groups

In ITA, MWTA, and STA the aggregation groups are defined over contiguous subsets of the non-temporal and timestamp domains. For the non-temporal attributes, each aggregation group is defined for a single attribute value; and for the timestamp, it is either divided into single time points, for ITA, or into contiguous sets of time points, for MWTA and STA.

It is desirable to also be able to compute aggregates over sets of argument tuples that are non-contiguous in some of the attributes. With general temporal aggregation, aggregation groups can be specified where the time domain is grouped into non-contiguous groups of time points and the timestamps of the tuples in an aggregation group do not necessarily overlap with the timestamp of the corresponding result tuple. Similarly, the aggregation groups need not be disjoint with respect to non-temporal attributes. We illustrate these capabilities by means of two examples.

Example 4. Consider the following query: *What is the total number of contracts in each quarter, summed up over the past two years*? In this query the argument tuples that contribute to a result tuple are temporally non-contiguous and do not overlap with the timestamp of the result tuple. This query can be formulated as $\mathcal{G}^T[\mathbf{g}, \theta, \mathbf{F}]\mathbf{emp}$, where:

$$\begin{aligned} \mathbf{g} &= \{([1,3]), ([4,6]), ([7,9]), ([10,12])\} \\ \theta(r) &= \{g \in \mathbf{g} \mid g.T \cap [\mathbf{emp}.T_s \bmod 12 + 1, \mathbf{emp}.T_e \bmod 12 + 1] \neq \emptyset\} \\ \mathbf{F} &= \{\mathit{count}_{CID}/\mathit{Cnt}\} \end{aligned}$$

The partial result tuples simply specify the four quarters, whereas the mapping function associates the argument tuples with the correct quarters.

Example 5. Consider the following query: *For each department, what is the total number of contracts in the other departments*? Here, the aggregation group of a partial result tuple consists of tuples with a department value that is different from the department of the partial result tuple. This query can be formulated as $\mathcal{G}^T[\mathbf{g}, \theta, \mathbf{F}]\mathbf{emp}$, where:

$$\begin{aligned} \mathbf{g} &= CI(\pi[\mathbf{r}.D, \mathbf{s}.T](\mathbf{r} \ltimes[\mathbf{r}.D \neq \mathbf{s}.D]\, \mathbf{r}/\mathbf{s})) \\ \theta(r) &= \{g \in \mathbf{g} \mid g.D \neq r.D \wedge g.T \cap \mathbf{emp}.T \neq \emptyset\} \\ \mathbf{F} &= \{\mathit{count}_{CID}/\mathit{Cnt}\} \end{aligned}$$

where CI is a regular expression that computes the constant intervals as for ITA. Note that the aggregation groups are not disjoint. With each partial result tuple we associate all argument tuples with a different department value.

5 Open Challenges in Temporal Aggregation

The foundations of most temporal database technology were built in the 1980s and 1990s. In retrospect, much of that research seems to have focused implicitly on meeting the relational data management needs of administrative applications. (This paper's example database is a good representative of this class of application.) Over the last decade, new types of applications and technologies have gained in prominence, including ones that offer new challenges to temporal database technology and temporal aggregation. We proceed to discuss challenges, most of which are due to these developments.

Update-Intensive Applications Based on Sampled Continuous Functions. The class of update-intensive applications is gaining in prominence. For example, large populations of vehicles may report their speeds and other sensed data. These data are samples of continuous functions. For most times, a measured value of a function is not available. The samples may have been reported according to a scheme that offers accuracy guarantees, or they may have been reported at regular time intervals. This is unlike the salary attribute in our example, and this scenario suggests several challenges.

First, we may want to transform the sequences of samples to a representation where we have a value for each point in time so that we are back in known territory. Issues include how to accomplish this transformation, how accurately to do this, and how to capture the inaccuracy.

Next, when applying an aggregation function to the sensed data, it becomes relevant to take into account the inaccuracy of the data so that the inaccuracy of the result can be reported. Likewise, when using the sensed data for defining the partial result relation, the inaccuracy of the data is an important part of the equation.

Third, it may be observed that instant temporal aggregation and moving-window temporal aggregation may return result relations that contain up to twice as many tuples as the input relations, which seems counter to the goal of summarizing the data in order to obtain an overview. It thus becomes of interest to be able to "aggregate an aggregate." We believe that it would be attractive to enable the users to control the trade-off between result accuracy and result cardinality. For example, if the user specifies a certain required accuracy, the aggregation should return the smallest number of tuples needed to satisfy that accuracy.

Applications Involving Higher-Dimensional Temporal Data. Many application will involve bitemporal, spatio-temporal, or n-dimensional data. Supporting aggregation for such data offers several challenges. For example, with more than one dimension, it becomes necessary to define the $1+$-dimensional equivalents of constant intervals. While constant intervals are unique, such constant regions are not. The definition as well as efficient implementation of maximal constant regions is a challenge.

Expressing General Temporal Aggregation in SQL. The SQL:2003 standard supports window functions. With these, aggregates may be computed by sorting and scanning the argument relation. While this is efficient, it does not support multidimensional groupings for which no single obvious ordering exists. Chatziantoniou's EMF-SQL extends the `group by` clause with grouping variables and introduces a `such that` clause for constraining the grouping variables [5]. Neither approach supports the specification of constant intervals, which is at the core of temporal aggregation. It would be interesting to extend these approaches with support for time. A survey of approaches to temporal aggregation in SQL-based temporal query languages is available [4].

Extension to Non-Relational Data Models. Far from all data is stored in SQL databases. Perhaps most notably, increasing amounts of data are stored in XML. Introducing temporal support, including support for temporal aggregation, calls for reconsidering many of the key data model and query language design decisions. For example: What is the equivalent of a tuple? Is there something comparable to tuple-timestamping and attribute-value timestamping? What are the implications of the hierarchical nature of the model for timestamping and aggregation?

Efficient Evaluation Algorithms. The general model covered in this paper defines temporal aggregation and offers a uniform way of expressing concisely the various forms of temporal aggregation that have been studied in the past. However, the definition does not imply an efficient implementation—a straightforward implementation would require costly operations such as joins and scans of the argument relation (4 scans for the delimiting points of the constant intervals and one for the aggregation). While efficient implementation of aggregation has been studied, solutions that integrate tightly with state-of-the-art relational database technology are in order. One specific challenge is to incrementally compute the partial result tuples as the argument relation is scanned, to avoid more than one scan of the argument relation.

6 Concluding Remarks

The framework for aggregation that has been available in SQL for several decades and that was formalized by Klug is very intuitive and has remained relatively unquestioned, at least in the context of on-line transaction processing. We believe that it is time to probe deeper. Specifically, the current framework is far from a panacea for all relational data management needs. We believe that aggregation can be rendered much more expressive.

This paper has elaborated on that view, by presenting a general framework for temporal aggregation, by illustrating how this framework accommodates existing forms of aggregation, and by pointing out new challenges that invite others to engage in further research—the general model proposed here is also not a panacea.

References

1. Ben-Zvi, J.: The Time Relational Model. Ph.D.thesis, Comp. Sci. Department, UCLA (1982)
2. Böhlen, M.H., Jensen, C.S., Snodgrass, R.T.: Temporal statement modifiers. ACM Transactions on Database Systems 25(4), 407–456 (2000)
3. Böhlen, M.H., Gamper, J., Jensen, C.S.: Multi-dimensional aggregation for temporal data. In: International Conference on Extending Database Technology, pp. 257–275 (2006)
4. Böhlen, M.H., Gamper, J., Jensen, C.S.: How would you like to aggregate your temporal data? In: Intl. Symposium on Temporal Representation and Reasoning, pp. 121–136 (2006)
5. Chatziantoniou, D.: Using grouping variables to express complex decision support queries. Data and Knowledge Engineering 61(1), 114–136 (2007)
6. Klug, A.C.: Equivalence of relational algebra and relational calculus query languages having aggregate functions. Journal of the ACM 29(3), 699–717 (1982)

7. Navathe, S.B., Ahmed, R.: A temporal relational model and a query language. Information Sciences 49(1-3), 147–175 (1989)
8. Snodgrass, R.T., Gomez, S., McKenzie, L.E.: Aggregates in the temporal query language TQuel. IEEE Transactions on Knowledge and Data Engineering 5(5), 826–842 (1993)
9. Snodgrass, R.T. (ed.): The TSQL2 Temporal Query Language. Kluwer, Dordrecht (1995)
10. Tansel, A.U.: A statistical interface to historical relational databases. In: International Conference on Data Engineering, pp. 538–546 (1987)
11. Vega Lopez, I.F., Snodgrass, R.T., Moon, B.: Spatiotemporal aggregate computation: a survey. IEEE Transactions on Knowledge and Data Engineering 17(2), 271–286 (2005)
12. Yang, J., Widom, J.: Incremental computation and maintenance of temporal aggregates. VLDB Journal 12(3), 262–283 (2003)