Noname manuscript No.
(will be inserted by the editor)

An implicit approach for periodic data in relational databases

Paolo Terenziani · Bela Stantic · Guido Governatori · Alessio Bottrighi · Abdul Sattar

Received: date / Accepted: date

Abstract Periodic data play a major role in many application domains, spanning from manufacturing to office automation, from scheduling to data broadcasting. In many of such domains, the huge number of repetitions make the goal of *explicitly* storing and accessing such data very challenging. In this paper, we propose a new methodology, based on an *implicit* representation of periodic data. The representation model we propose captures the notion of periodic granularity provided by the temporal database glossary, and is a consistent extension of the TSQL2 temporal relational data model. On top of our new data model, we propose a suitable indexing technique. We define the algebraic operators, and introduce access algorithms to cope with them, proving that they are correct and complete with respect to the traditional explicit approach. We also propose an experimental evaluation of our approach.

Keywords temporal databases · periodic data · implicit representation

1 Introduction

Periodic events seem to be an intrinsic part of our life, and of our way of perceiving reality. Day and nights repeat at regular periodic patterns, as well as seasons,

P. Terenziani, A. Bottrighi
Department of Computer Science, University of Piemonte Orientale
"Amedeo Avogadro", Alessandria, Italy
Tel.: +39-0131-360174
Fax: +39-0131-360198
E-mail: terenz@mfn.unipmn.it, alessio@di.unipmn.it

B. Stantic, A. Sattar
Institute for Integrated and Intelligent Systems Griffith University, Brisbane Australia

G. Governatori
NICTA, Brisbane Australia

and years. Accordingly, many human activities are scheduled at periodic time (consider, e.g., office activities, scheduling of train, airplanes, lessons, ...). Periodic data play a major role in many application domains, spanning from financial trading to scheduling, from manufacturing and process control to office automation and data broadcasting. Due also to such a wide range of different contexts of application, it is widely agreed that adopting a "standard" and fixed menu of granularities (e.g., minutes, hours, days, weeks, years and so on in the Gregorian calendric system) is not enough in order to provide the required expressiveness and flexibility. For instance, Soo and Snodgrass (1993) emphasized that the use of a calendar depends on the cultural, legal and even business orientation of the users, listed many examples of different calendric systems and user-defined periodic granularities (e.g., the academic vs legal vs financial year) and stressed that different user-defined periodic granularities are usually used even in the same area (consider, e.g., the definition of holidays in different companies, or in different school institutions). Moreover, the number of repetitions of periodic data may be very large (in some cases, repetitions may also be "open-ended"-e.g., therapies for chronical patients may be repeated all the life long). Therefore, in the Computer Science literature (and, in particular, in the areas of Databases, Logics, and Artificial Intelligence), there is a common agreement that *formalisms* are needed in order to cope with *user-defined* periodic data in an *implicit* (elsewhere termed *intensional*) way (i.e., without an explicit storing of all the repetitions; see also the discussion in 2.1), and a large number of approaches has been defined to such a purpose (e.g., the survey by Tuzhilin and Clifford (1995), dating back to 1995, focuses on the Database area, and takes into account 34 different approaches). Periodic data[1] play an important role in Databases, so that, for instance, a specific entry (see Terenziani (2009)) has been devoted to such a topic in the outcoming Encyclopedia of Database Systems by Springer Liu and Tamer zsu (2009.)). In the Encyclopedia Terenziani (2009), three main classes of Database (implicit) approaches to user-defined periodicities have been identified: *Deductive rule-based* approaches, using deductive rules (e.g., Chomicki and Imielinski (1993) and approaches in classical temporal logics), *constraint-based* approaches, using mathematical formulae and constraints (e.g., Kabanza et al (1995)), and *algebraic* (also termed *symbolic*) approaches, providing a set of "high-level" and "user-friendly" operators (e.g., Leban et al (1986), Niezette and Stevenne (1992), Bettini and De Sibi (2000), Ning et al (2002), Terenziani (2003), Egidi and Terenziani (2005), Anselma et al (2006), Terenziani (2000), Egidi and Terenziani (2004), Egidi and Terenziani (2006), Egidi and Terenziani (2008)). A comparison among such classes approaches is out of the scope of this paper (the interested reader is referred to Terenziani (2009) and also to Egidi and Terenziani (2006)). However it is worth stressing that, in most approaches in the literature (and, in particular, in all algebraic approaches), the focus is on the design of *high-level formalisms* to model (in an implicit way) user-defined periodicities in a "commonsense" or at least "user-friendly" way. Most of such approaches do not

[1] It is worth mentioning that, according to the temporal database literature, we term periodic those data that have value-equivalent repetitions at periodic time (e.g., the periodic schedule of trains); data that are acquired at periodic time, but may assume different values (e.g., periodic monitoring of blood pressure) are not taken into account in this paper, as well as in all the approaches to periodic data referenced in this paper.

take into account issues such as the definition of *relational temporal algebraic operators*, extending Codd's operators to query periodic data, *range queries* and efficient *indexing* and *access* of temporal data. Additionally, to the best of our knowledge, the algebraic (symbolic) approaches that have extended Codd's operators (i.e., Kabanza et al (1995), Niezette and Stevenne (1992), Terenziani (2003)), have provided temporal algebraic operators whose complexity is exponential (see the discussion in Section 6). In summary, although there seems to be a general agreement within the Database (and also Artificial Intelligence) literature that general-purpose implicit approaches are needed in order to cope with *user-defined* periodic data, and despite the fact that a lot of such approaches have been devised in the last two decades, none of such approaches focus specifically on the definition of a comprehensive relational approach coping with user-defined periodic data efficiently, considering both a relational representation of periodic data and algebraic operators operating on it.

However, all such issues are fundamental for the practical applicability of any Database approach considering periodic data. The goal of our work is that of devising such a comprehensive approach, in a "principled way". Specifically, our approach has been designed in such a way that

(i) our data model has the expressiveness to capture all "periodic granularities", as defined in the Database literature Bettini et al (1998), Bettini and De Sibi (2000),
(ii) our data model is a consistent extension of the TSQL2 consensus model Snodgrass (1995), and
(iii) our algebraic and temporal query operators are correct and complete with respect to conventional explicit approaches, in which all the repetitions of periodic data are explicitly stored. Additionally, our extended algebraic operators operate in polynomial time, and are a consistent extension of standard nontemporal relational algebraic ones.

Property (i) grants that the expressiveness of our data model is the one requested by the temporal Database literature. On the other hand, property (ii) grants that our approach can be added on top of TSQL2 as a support to cope with periodic data. In turn, it is worth noticing that TSQL2 has been proven to be a consistent extension of the standard relational model, and can be reduced to it in case time is disregarded. Therefore, property (ii) is essential, since it grants the interoperability of our approach with pre-existent TSQL2 or standard relational data. Finally, property (iii) grants that, although periodic data are only implicitly stored, we get the same (correct) results obtained with traditional (i.e., fully explicit) models. Moreover, in this paper we also provide testing, to show the advantages of our approach with respect to conventional explicit approaches, especially in terms of disk I/O's.

On the other hand, in this paper:

- We do not address the treatment of the transaction time of events (i.e., the time when events are 'inserted in'/'deleted from' the database Snodgrass and Ahn (1985)). As a matter of fact, transaction time is *orthogonal* to valid time (i.e., the time when the fact described by the tuples takes place), and no periodicity issue is usually involved by it. As a consequence, the approach dealing with the (periodic) valid time proposed in this paper can be trivially extended to deal with

also with transaction time, by coping with transaction time in the standard way proposed in the temporal database literature.
– Although in this paper we cope with *user-defined* periodic granularities we assume that each periodic granularity is directly expressed in terms of a "bottom" granularity (e.g., "seconds"; as we will see, this is not a limitation, given the definition of periodic granularity). Therefore, in this paper, we are not interested to cope with issues concerning, e.g., *conversions* between periodic granularities, or *properties of relations between them* (except that to the bottom one), which is, on the other hand, a main focus of other approaches dealing with multiple (possibly periodic) granularities (consider, e.g., Snodgrass (1995), Ning et al (2002), Dyreson et al (1995), Bettini et al (2000)).

The rest of the paper is organized as follows. Section 2 is a preliminary one, in which we first briefly mention the implicit vs explicit dichotomy, and then we report the basic definitions of periodic and quasi-periodic temporal granularities Bettini and De Sibi (2000)(where quasi-periodic granularities extend periodic granularities to cope with *finite* exceptions). In Section 3 we propose an extended relational temporal data model coping with quasi-periodic granularities. In section 4 we extend the definition of Codd's algebraic operators to cope with our temporal data model. In section 5, we present an experimental evaluation of our approach, showing its advantages with respect to the "traditional" extensional approach. Finally, section 6 addresses conclusions, comparisons and future work.

2 Preliminaries

In order to settle the stage, in this section we first report some of the main arguments in the literature in favor of implicit approaches with respect to explicit ones. We then introduce the basic definitions in the Database literature on which our approach is grounded.

2.1 Implicit vs. explicit approaches to periodic data

There is an obvious and trivial way to cope with (possibly user-defined) periodic data, namely by explicitly storing all of them. Such an approach, usually called "explicit" (or "extensional") approach, basically reduces periodic data to standard non-periodic ones. For instance, in order to deal with an activity X scheduled each day from 10 to 12 a.m. in a year, an explicit approach can simply represent the *valid time* of the activity through list all of the 365 time *periods* in which the activity takes place (see the temporal database glossary as regards the terminology we adopt Jensen and et al. (1998)). The obvious advantage of such an approach is its simplicity: periodic temporal data are simply coped with as standard temporal data, so that any temporal Database approach in the literature can suffice. Moreover, it makes all of the database implementation simpler, from indexing to query processing. However, there are at least three main disadvantages to the "explicit" approach:

- it is not "commonsense" and "human-oriented": humans usually tend to abstract, so that they usually prefer to manage periodic data in an implicit way. For instance, in the aforementioned example, the list of 365 time periods is not probably the most user-friendly and perspicuous answer to a user wanting to know when activity X must be performed (an "implicit" answer such as "all days from 10 to 12 a.m." is, in most cases, a more desirable answer).
- it is very expensive in term of physical disk I/O's, due to the high storage size. In many practical applications areas, the number of repetitions (at periodic time) of activities is very high, so that making explicit all the repetitions is very space-demanding. For instance, bus or train scheduling repeats regularly every day (or, in some cases, every week) for long intervals of time (a season, or a year). Recording the activities on an automatized schedule in a production chain is usually even more space-demanding. In fact, in this application domain, the same activity may be performed at a very high (periodic) frequency (even several times each minute) for very long overall frames of time (production can go on for years). Making all such data explicit might rapidly reach a critical data size even for the most efficient commercial DBMS, with dramatic consequences especially in term of physical disk I/O's.
- it is not feasible in the case of "open ended" data (i.e., of data whose valid time is open in the future, and for which there is no known future end). Dealing with open ended data one does not know the end point of repetitions, so that no explicit elicitation of all the data is possible.

2.2 (quasi)-periodic granularities

In this preliminary section, we give the definition of granularity taken from the temporal database glossary Bettini et al (1998) (see Definitions 1 to 4 below), and its successive extension to cover periodic and quasi-periodic temporal granularities Bettini and De Sibi (2000) (see Definition 5 below). Such definitions are the basis for our treatment of periodic data (i.e., data whose validity time can be described by user-defined periodic granularities).

Definition 1 (Time domain) A time domain is a pair $(T, \leq)$, where T is a non-empty set of time instants and $\leq$ is a total order on T.

The time domain can be be $(Z, \leq)$, $(N, \leq)$, or $(Q, \leq)$.

Definition 2 (Granularity) A granularity is a mapping G from the integers (the index set) to subsets of the general time domain such that:

(i) if $i < j$ and $G(i)$ and $G(j)$ are not empty, then each element of $G(i)$ is less than all elements of $G(j)$, and
(ii) if $i < k < j$ and $G(i)$ and $G(j)$ are not empty, then $G(k)$ is not empty.

Basically, condition (i) grants that the granules in a granularity do not overlap in time, and that their indexes are ordered consistently with the time domain; Condition (ii) states that the elements of the index domain that map onto non-empty subsets of the time domain are contiguous.

Definition 3 (Granule) Each nonempty subset of the time domain in the image of a granularity is called granule.

Granules have a specific topology induced by the granularity function. In particular, the granularity defines a distinguished origin granule, e.g., $G(0)$. Granularities provide a formal representation of abstract calendric concepts. In this paper, we restrict our attention to periodic and quasi-periodic granularities. The formal definition requires the definition of some relationships between granularities.

Definition 4 (groups into) A granularity G **groups into** a granularity H, if for each index j of H there exists a subset S of the integers such that $H(j) = \bigcup_{i \in S} G(i)$.

Intuitively, G groups into H if each granule of H is the union of a set of granules of G (e.g., days groups into weeks). Periodic and quasi-periodic granularities can now be defined. Such definitions will be commented and revisited in the next section, where we will provide our formalism to cope with quasi-periodic granularities in a relational context.

Definition 5 (periodically groups into) A granularity G **periodically groups into** a granularity H if

(i) G groups into H, and
(ii) there exist positive integers n and m, where n is less than the number of nonempty granules of H, such that for all $i \in Z$, if $H(i) = \bigcup_{r=0}^{k} G(j_r)$ and $H(i+n) \neq \emptyset$, then $H(i+n) = \bigcup_{r=0}^{k} G(j_r + m)$.

Intuitively, the quasi-periodic groups-into relation is basically a periodic *groups-into* relation, but with some "additional granules". It should be stressed that the discussions before and after the definition of a quasi-periodic grouping in Bettini and De Sibi (2000) imply that the set of additional granules is finite and that each addition is a finite period; therefore we have added these constraints in the definition below.

Definition 6 (quasi-periodically groups into) A granularity G **quasi-periodically groups into** a granularity H if

(i) G groups into H, and
(ii) there exists a finite set of finite intervals $E_1, \ldots, E_z$ (the granularity exceptions) and positive integers n and m, where n is less than the minimum of the number of granules of H between any two exceptions, such that for all $i \in Z$, if $H(i) = \bigcup_{r=0}^{k} G(j_r)$ and $H(i+n) \neq \emptyset$, and $i+n < min(E)$, where E is the closest existing exception after $H(i)$ (if such exception exists; otherwise $E = max(k|H(k) \neq \emptyset)$, then $H(i+n) = \bigcup_{r=0}^{k} G(j_r + m)$.

Finally, the definition of periodic and quasi-periodic granularities is given in Bettini and De Sibi (2000) in terms of a bottom granularity.

Definition 7 (Quasi-periodic granularity) A periodic (resp. quasi-periodic) granularity is a granularity periodic (resp. quasi-periodic) with respect to the bottom granularity.

3 Representing (quasi)-periodic granularities

In this section, we propose a representation of (quasi)-periodic granularities, based on the above general definitions. Our implicit representation of a quasi-periodic granularity G is a quadruple:

Definition 8 In our approach a quasi-periodic granularity G is represented by a quadruple

$$G = \langle P, I_P, I_E, FT \rangle$$

where P is an integer representing the duration of the periodic pattern; I_P is the set of the convex periods in the first "periodic pattern" (after the granule "0" of the bottom granularity); I_E is the set of the convex periods constituting the aperiodic part; FT is the "frame time" i.e., the period containing all rhe repetitions of the periodic pattern. In turn, a period having as first granule the bottom granule $B(i)$ and as last granule the bottom granule $B(j)$ is represented by "$[i, j]$".

Let us exemplify our representation through an example.

Example 1 As a working example, let us consider the following user-defined granularity. Let us suppose that, in the year 2007, starting from Monday January 8th and ending on Sunday December 23rd (here we assume that weeks start on Monday), the "working shift" for a company is from 08:00 to 12:00, and from 14:00 to 18:00 each day from Monday to Friday (considering each day as a unique period, possibly with gaps in it), and from 8 to 12 on Saturday (let us call such a granularity "WS"). In addition, let us suppose that he also works on Saturday evening (from 14 to 18) in two specific days (say on January 13 and 20; let us call "WS+" the granularity WS with such an addition).

WS and WS+ are represented in our formalism as follows, considering hours as the bottom granularity (notice that our approach is independent of the choice of the bottom granularity).

Example 2

$$\begin{aligned} \mathrm{WS} &= \langle 168, \{[176,179],[182,185],[200,203],\ldots,[296,299]\}, \\ &\quad \{\}, [168,8567] \rangle \\ \mathrm{WS+} &= \langle 168, \{[176,179],[182,185],[200,203],\ldots,[296,299]\}, \\ &\quad \{[2029,2033],[2197,2201]\}, [168,8567] \rangle \end{aligned}$$

In particular, 168 hours (one week) is the duration of the periodic pattern, $\{[176, 179], [182, 185], [200, 203], \ldots, [296, 299]\}$ is the first instance of the periodic pattern, $\{[2029, 2033], [2197, 2201]\}$ are the two additional repetitions for WS+ (the set of additional repetitions is empty in WS), and $[168, 8567]$ is the frame of time containing all the periodical repetitions on the periodic pattern.

Notice that, for the sake of generality, we admit that the non-periodic repetitions (if any) in a quasi-periodic granularity are outside the frame time. In other words,

we regards the frame time as the span of time in which data repeats regularly (while additions are kept separate).

Before analyzing how such an abstract representation can be modelled in the relational context, let us briefly sketch the data model of TSQL2, probably the largest attempt to establish a "consensus" approach within the temporal databases community Snodgrass (1995)). Then we extend it to cope (in an implicit way) with periodic data, discussing the relationships between our extended model and TSQL2's one.

3.1 TSQL2 data model

In general, (temporal) databases are used to store both the non-temporal data (in the form of tuples belonging to relations, if the relational model is used) and the temporal aspects (e.g., the valid times) concerning it. In many approaches (and, in particular, in TSQL2), valid time is associated to relational tuples, in the form of a pair of timestamps (the first denoting the starting point of the valid time, and the second its ending point).

Definition 9 (TSQL2 valid time relation). Given any schema $R = (A_1, \ldots, A_n)$ (where $A_1, \ldots, A_n$ are standard non-temporal attributes), a valid time relation r in TSQL2 is a relation defined over the schema $R^V = (A_1, \ldots, A_n \mid VT_S, VT_E)$ where VT_S and VT_E are timestamps representing the starting and the ending time of the valid time period respectively.

In this paper, we propose a generalization of such an approach, in order to cope also, in a "implicit" way, with "quasi-periodic" data.

Definition 10 (Quasi-periodic data). In the following, we use the term "quasi-periodic" data (tuple) to refer to data (tuples) holding at periodic valid times (i.e., to data holding on (quasi-) periodic granularities).

3.2 A relational representation of quasi-periodic data

The abstract representation of quasi-periodic granularities above is the basis to define our extended model, coping with quasi-periodic data in a relational environment. However, several aspects need to be investigated, and choices done. For instance, we could associate a unique identifier to each user-defined quasi-periodic granularity, and extend the data model with just an additional attribute, used in order to pair each tuple with the identifier of the granularity. One (or more) dedicated tables could then be used in order to associate with each identifier the "implicit" description of the granularity they denote. Although quite simple, such a solution presents several drawbacks.[2] In particular, from the theoretical point of view, such a solution

[2] For instance, from the technical point of view, it is not a very efficient implementation, since the association of each tuple with any information concerning its valid time (which is supposed to be a quite frequent operation) requires a "join" operations between different relations (tables).

is not a "consistent extension" of the usual representation of valid time (e.g., the one adopted in TSQL2), in which, as sketched above, valid time is represented by a pair of timestamps. Devising a "consistent extension" of the model used in a "consensus" approach such as TSQL2 is one of the main desiderata of our approach, since it guarantees that the "consensus" approach can be seen as a subcase of our general framework, so that it can be still used in order to deal with simple (i.e., non-periodic) cases. Moreover, our model is also based on the two considerations below:

- Given a periodic tuple, its "frame time" can be interpreted, roughly speaking, as a rough approximation of its "valid time", in the sense that it contains all the time periods in which the tuple holds.
- Given a quasi-periodic tuple, the "non-periodic" part of its granularity can be simply represented by a set of time periods, i.e., of "standard" valid times in the "consensus" approach.

We can now define our new data model (called "periodic" data model) as follows:

Definition 11 (periodic relation). Given any schema $R = (A_1, \ldots, A_n)$ (where $A_1, \ldots, A_n$ are standard non-temporal attributes), a periodic relation r is a relation defined over the schema $R^P = (A_1, \ldots, A_n \mid VT_S, VT_E, Per, Per_{id})$ where

- VT_S is a timestamp representing the starting point of the "frame time"
- VT_E is a timestamp representing the ending point of the "frame time"
- Per is an interval, representing the duration of the repetition pattern
- Per_{id} is an identifier, denoting a periodic pattern

In addition, in order to code periodic patterns, an additional dedicated relation (a valid-time relation, in the sense of TSQL2) is needed (called `Periodicity` relation henceforth).

Definition 12 (Periodicity relation). The periodicity relation `Periodicity` is a relation over the schema $(Periodicity\_ID, Start, End)$, in which

- $Periodicity\_ID$ is a textual attribute containing identifiers denoting periodic patterns
- $Start$ and End are temporal attributes (timestamps) denoting the starting and the ending points of the periods in the periodic pattern.

It is importante to notice that, by construction, the temporal attributes of our periodic relations, in conjunction with the Periodicity relation, allow us to capture the implicit definitions of periodic and quasi-periodic granularities, so that the following property holds:

Property 1 (Expressiveness) Our extended relational data model can represent periodic and quasi-periodic granularities, as defined in Jensen and et al. (1998).

Moreover, it is worth noticing that the non-periodic part of quasi-periodic data can be easily represented as a degenerate case of the periodic one. In fact, consider a quasi-periodic tuple t belonging to a periodic relation r, and having as aperiodic part a set of the convex periods $p_1, \ldots, p_k$. Its aperiodic part can be simply coded in r, by using

k tuples value-equivalent Snodgrass (1995) to t (i.e., having the same values as t as concerns the non-temporal values), each one having as VT_S and VT_E the starting and ending point of one of the periods (i.e., one of $\{p_1, \ldots, p_k\}$), and $NULL$ values for the other temporal attributes (i.e., *Per*, and *Per_id* are set to $NULL$;[3].).

As a consequence, a database in our model can be defined as follows:

Definition 13 (database) In our extended model, a database is a set of periodic relations (see definition 11), plus a dedicated `periodicity` relation.

Notice that, for the sake of efficiency, in our approach a database can also contain valid-time relations and standard non-temporal relations (even if this is not strictly necessary, from the theoretical point of view; see the discussion in the subsection below).[4] However, since our treatment of such relations is the standard one, in the rest of this paper, we just focus on periodic relations.

As an example, let us suppose to consider a periodic relation `Activity`, coping with the activities of the employees of an office. Each activity is simply represented by an activity identifier (attribute *ActID*), a textual descriptor of the activity type (attribute *Act*) and by the (identifier of the) employee who has to perform it (attribute *ActorID*), plus the temporal part. As an example, let us suppose that 'John' has to perform activities of type 'A' at the quasi-periodic time $WS+$ defined above. The corresponding representation in our extended relational model is shown in Tables 1 and 2.

3.3 Relations with TSQL2 data model

It is interesting to analyze the relationships between our extended model and the TSQL2 data model (which, in turn, is an upward compatible extension of the standard SQL non-temporal one Snodgrass (1995)).

Actually, a TSQL2 valid-time relation can be seen as a degenerate case of a periodic relation in our approach, in which:

- the valid time corresponds to the frame time
- the (periodic) pattern exactly covers the whole frame time

In other words, we may interpret a TSQL2 valid-time tuple starting at VT_S and ending at VT_E as a degenerate tuple of a periodic relation, having as frame time the period starting at VT_S and ending at VT_E. Such a tuple is degenerate, in the sense that its periodic pattern covers exactly the frame time (i.e., there is exactly one repetition of the tuple, holding exactly on the whole frame time).

As a consequence, valid-time TSQL2 relations can be modeled (although not efficiently) as periodic relations having NULL values for the P and P_{id} attributes.

As a desirable side effect, we can easily code non-periodic valid-time tuples in our model, as periodic tuples having NULL values for the *Per* and Per_{id} attributes.

[3] In order to avoid to overload the NULL value, we could also choose to represent aperiodic tuples by specifying *Per* as the duration of their valid time, and *Per_id* as a pre-defined default value

[4] Moreover, we can also deal with transaction time Snodgrass (1995).

ActID	*Act*	*ActorID*	VT_S	VT_E	*Per*	*PerID*
1	*A*	*John*	168	8567	168	*P*1
2	*A*	*John*	2029	2033	*Null*	*Null*
3	*A*	*John*	2197	2201	*Null*	*Null*

Table 1 `Activity` periodic relation – Implicit model

Pattern ID	*Start*	*End*
*P*1	176	179
...	...	...
*P*1	296	299

Table 2 `Periodicity` relation – Implicit model

ActID	*Act*	*ActorID*	VT_S	VT_E
1	*A*	*John*	176	179
2	*A*	*John*	182	185
3	*A*	*John*	200	203
4	*A*	*John*	...	...
...	*A*	*John*	296	299
...	*A*	*John*	...	...
...	*A*	*John*	8627	8631

Table 3 `Activity_Expl` relation – Explicit model

Therefore, standard TSQL2 tuples (and relations) can be modelled in our approach. In this sense, we can say that the following property holds:

Property 2 (consistent extension) Our data model is a "consistent extension" of TSQL2 data model.

3.4 Explicit representation of periodic data

In the experimental part of this paper, we will use range queries to compare our approach based on implicit modelling with the explicit approach. In the following, we show an explicit (TSQL2-style) representation corresponding to the implicit representation (called `Activity_Expl`) in relation `Activity` above. Table 3 explicitly represents a part of the data implicitly represented in the Tables 1 and 2 above.

4 Temporal algebra

Codd designated as complete any query language that was as expressive as his set of five relational algebraic operators, relational union ($\cup$), relational difference ($-$), selection (σ_P), projection (π), and Cartesian product ($\times$) Codd (1971). In this section we propose an extension of Codd's algebraic operators in order to query the new data model introduced in section 3. Such a temporal algebra can provide the ground for

defining a proper extension of SQL, or of the temporal language TSQL2 Snodgrass (1995), to cope with periodic data, which is the goal of our future work.

Several temporal extensions have been provided to Codd's operators in the temporal database literature Snodgrass (1995), L. Edwin McKenzie and Snodgrass (1991). In many cases, such extensions behave as standard non-temporal operators on the non-temporal attributes, and involve the application of set operators on the temporal parts. This approach ensures that the temporal algebrae are a consistent extensions of Codd's operators and are reducible to them when the temporal dimension is removed. For instance, in BCDM Jensen and Snodgrass (1996), which provides a uniform semantics underlying several temporal database approaches, including the "consensus" approach TSQL2 Snodgrass (1995), temporal Cartesian product involves pairwise concatenation of the values for non-temporal attributes of tuples and pairwise intersection of their temporal values.

We ground our approach on such a "consensus" background, extending it in order to cope with periodic data. Before proceeding to the definition, we need to introduce a brief digression about the treatment of value-equivalent tuples. In the temporal relational literature, two tuples are said to be $value-equivalent$ if they have exactly the same values as regards their non-temporal attributes. In the BCDM model, Jensen and Snodgrass (1996), it is agreed that (from the abstract -semantic- point of view) each tuple should be equipped with all its temporal information, so that no value-equivalent tuple can coexist in the same relation. While this is a major source of clarity for the abstract model, the different practical logical representations adopted in the literature have used different strategies to cope with value-equivalent tuples (see, e.g., the discussion in the TSQL2 book Snodgrass (1995)). For instance, the "consensus" TSQL2 approach admit value-equivalent tuples *at the logical (representation) level*, still retaining the underlying semantics dictated by the BCDM model. This choice has a strong impact on the definition of relational algebraic operators. For instance, in BCDM, and in the logical representations in which no value-equivalent tuples are admitted, relational union need to *coalesce* Böhlen et al (1996) the times of value equivalent tuples deriving from the relations being united; on the other hand, if value-equivalent tuples are admitted *at the representation level*, temporal relational union can simply put all the input tuples in the result. However, in order to maintain the underlying BCDM *semantics*, in such approaches (such as TSQL2), temporal relational difference need to consider the fact that value-equivalent tuples may be present in both input relations, so that all their times must be collected before performing the temporal difference between the temporal components of the tuples.

In the following, we have chosen to follow the line of TSQL2 "consensus" approach, thus admitting value-equivalent tuples in our model, but our approach is mostly independent of such a choice.

In the definitions below, we denote by $t[X_1,\ldots,X_k]$ the value of the attributes $X_1,\ldots,X_k$ in the tuple t.

Our temporal relational union takes in input the tuples of two input periodic relations r and s, and gives them in output unchanged both in the non-temporal and temporal part.

Definition 14 (Temporal union $\cup^T$) Given two periodic relations r and s defined over the schema $R^P = (A_1,\ldots,A_n \mid VT_S, VT_E, Per, Per_{id})$, the temporal union $r \cup^T s$ is a periodic relation q defined over the schema R^P, and is defined as follows:

$q = r \cup^T s = \{t \mid \exists t_r \in r, t[A_1,\ldots,A_n] = t_r[A_1,\ldots,A_n] \wedge$
$t[VT_S] = t_r[VT_S] \wedge t[VT_E] = t_r[VT_E] \wedge t[Per] = t_r[Per] \wedge t[Per_{id}] = t_r[Per_{id}] \vee$
$\exists t_s \in s\, t[A_1,\ldots,A_n] = t_s[A_1,\ldots,A_n] \wedge$
$t[VT_S] = t_s[VT_S] \wedge t[VT_E] = t_s[VT_E] \wedge t[Per] = t_s[Per] \wedge t[Per_{id}] = t_s[Per_{id}]\}$

Our temporal relational projection simply operate on the non-temporal part of the input tuples, retaining only the values of the input attributes. The temporal component of the tuples is left unchanged.

Definition 15 (Temporal projection $\pi^T_{A_i,\ldots,A_j}$) Given a periodic relation r defined over the schema $R^P = (A_1,\ldots,A_n \mid VT_S, VT_E, Per, Per_{id})$, and given a subset $\{A_i,\ldots,A_j\}$ of the set $\{A_1,\ldots,A_n\}$, temporal projection $\pi^T_{A_i,\ldots,A_j}(r)$ is a periodic relation q defined over the schema $R'^P = (A_i,\ldots,A_j \mid VT_S, VT_E, Per, Per_{id})$, and is defined as follows:

$q = \pi^T_{A_i,\ldots,A_j}(r) = \{t \mid \exists t' \in r, t[A_i,\ldots,A_j] = t'[A_i,\ldots,A_j] \wedge$
$t[VT_S] = t'[VT_S] \wedge t[VT_E] = t'[VT_E] \wedge t[Per] = t'[Per] \wedge t[Per_{id}] = t'[Per_{id}]\}$

The definition of selection on non-temporal attributes is trivial: only the input tuples whose non-temporal component satisfy the selection predicate ϕ are reported in output, unchanged (both in their temporal and nontemporal parts). Notice that ϕ is a predicate regarding non-temporal attributes only.

Definition 16 (Nontemporal selection σ^T_ϕ) Given a periodic relation r defined over the schema $R^P = (A_1,\ldots,A_n \mid VT_S, VT_E, Per, Per_{id})$, and a predicate ϕ regarding the non-temporal attributes only, $\sigma^T_\phi(r)$ is a periodic relation q defined over the schema R^P, and is defined as follows:

$q = \sigma^T_\phi(r) = \{t \mid \exists t' \in r, \phi(t[A_1,\ldots,A_n]) \wedge t[A_1,\ldots,A_n] = t'[A_1,\ldots,A_n] \wedge t[VT_S] = t'[VT_S] \wedge$
$t[VT_E] = t'[VT_E] \wedge t[Per] = t'[Per] \wedge t[Per_{id}] = t'[Per_{id}]\}$

In the definition of Cartesian Product, *lcm*, *min*, and *max* denote the least common multiple, minimum and maximum functions; *generate_id*() is a function that generated a new unique identifier; *pattern*(*id*,`Periodicity`) denotes the set of periods corresponding to the identifier *id* in the table `Periodicity`. Moreover, the function *generate_inters_pattern* takes in input two identifiers id_1 and id_2 of periodic patterns in the `Periodicity` table, the identifier id_{new} of a new pattern (the intersection pattern) to be introduced in the table, the time t_{start} at which the new pattern starts, its duration *Per*, and the `Periodicity` table. It operates in three steps

(1) It retrieves from the `Periodicity` table the patterns corresponding to id_1 and id_2 and makes such patterns explicit on the whole period $[t_{start}, t_{start} + Per]$; let $< p_{11},\ldots,p_{1k} >$ and $< p_{21},\ldots,p_{2h} >$ be the lists of temporally ordered periods obtained by making explicit the periodic pattern corresponding to id_1 and id_2 respectively (the function *make_explicit* accomplishes such tasks, and is described in the Appendix).

(2) It evaluates the intersection between $< p_{11},\ldots,p_{1k} >$ and $< p_{21},\ldots,p_{2h} >$; let $< p_1,\ldots,p_j >$ be the resulting ordered lists of periods; (see the function $g\_intersects$ below).
(3) If $< p_1,\ldots,p_j >$ is empty, the empty set is given as result, otherwise the table `Periodicity` is updated with the insertion of the new pattern $< p_1,\ldots,p_j >$ for the new identifier id_{new}.

Definition 17 (Temporal Cartesian product $\times^T$) Given two periodic relations r and s defined over the schemas $R1^P = (A_1,\ldots,A_n \mid VT_S, VT_E, Per, Per_{id})$ and $R2^P = (B_1,\ldots,B_k \mid VT_S, VT_E, Per, Per_{id})$ respectively, the temporal Cartesian product $r \times^T s$ is a periodic relation q defined over the schema $R3^P = (A_1,\ldots,A_n,B_1,\ldots,B_k \mid VT_S, VT_E, Per, Per_{id})$ and is defined as follows:

$q = r \times^T s = \{t \mid \exists t_r \in r, \exists t_s \in s,$
$t_r[Per] \neq NULL \wedge t_s[Per] \neq NULL \wedge$
$t[A_1,\ldots,A_n] = t_r[A_1,\ldots,A_n] \wedge t[B_1,\ldots,B_k] = t_s[B_1,\ldots,B_k] \wedge$
$t[VT_S] = max(t_r[VT_S], t_s[VT_S]) \wedge t[VT_E] = min(t_r[VT_E], t_s[VT_E]) \wedge t[VT_S] < t[VT_E] \wedge$
$t[Per] = lcm(t_r[Per], t_s[Per]) \wedge new_{id} = generate\_id() \wedge t[Per_{id}] = new_{id} \wedge$
$pattern(new_{id}, t[VT_S], t[Per], \texttt{Periodicity}) \neq \emptyset$ where
$pattern(new_{id}, t[VT_S], t[Per], \texttt{Periodicity}) =$
$generate\_inters\_pattern(t_r[Per_{id}], t_s[Per_{id}], new_{id}, t[VT_S], t[Per], \texttt{Periodicity})$

For each pair of tuples (one from r and one from s) the output is a tuple which (as discussed above) has as non-temporal part the concatenation of the two non-temporal parts, and as temporal part the intersection of the temporal parts. In our representation of periodic data, the intersection is obtained by (i) intersecting the frame times and (ii) intersecting the periodic pattern over a period of time which starts at the intersection of the frame times, and whose duration is the least common multiple of the duration of the two input patterns.

The definition of temporal Cartesian product given above can be extended to temporal definitions of theta join, natural join, outer joins, and outer Cartesian products, in a way similar that done in Gao et al (2005)[5].

While the choice of admitting value-equivalent tuples make the definition of the above operators quite easy (and efficient, since no manipulation on the temporal components is needed), the definition of temporal difference results necessarily to be quite complex, since an unpredictable number of value-equivalent tuples may be present in the input relations. Intuitively speaking, in the temporal difference $r -^T s$, each tuple $t \in r$ which has no value-equivalent tuple in s is simply reported unchanged in output. Otherwise, let $\{t_1,\ldots,t_k\}$ the set of all and only the tuples in s that are value-equivalent to t (the quantifier $\exists!$ is used in the definition to denote that they are "all and only"). The frame time of t must be "fragmented" into sub-periods, according to the frame times of $\{t_1,\ldots,t_k\}$ intersecting it. In the

[5] Also, notice that an "optimization" of the above definition is also possible. For the sake of clarity and generality, in the definition we model the result of the Cartesian product of two periodic tuples t_r and t_s as a periodic tuple t even in case the intersection of the frame times of t_r and t_s is shorter than the duration of the output periodic pattern (i.e., $t[VT_E] - t[VT_S] < t[Per]$), that is, even in case there is not at least one whole repetition of the resulting periodic pattern. The definition can be easily emended to distinguish such a case, modelling the result as a set of non-periodic tuples.

definition, the function $fragments$ is used for such a purpose (see the explanation below). For each one of such "fragments", all and only the tuples in $\{t_1,\ldots,t_k\}$ covering it must be considered. Let $\{t'_1,\ldots,t'_s\}$ be all and only such tuples. For each one of the "fragments" a tuple value-equivalent to t must be provided as output, having as duration the least common multiple of the durations of t and $\{t'_1,\ldots,t'_s\}$, and as periodic pattern the pattern obtained by making the difference (using the function $generate\_difference\_pattern$; see comments below) between the pattern of t and the union (using the function $generate\_union\_pattern$; see comments below) of the patterns of $\{t'_1,\ldots,t'_s\}$. Of course, if the resulting pattern is empty, no tuple must be provided in output for the corresponding "fragment".

Definition 18 (Temporal difference $-^T$) Given two periodic relations r and s defined over the schema $R^P = (A_1,\ldots,A_n \mid VT_S, VT_E, Per, Per_{id})$, the temporal difference $r -^T s$ is a periodic relation q defined over the schema R^P defined as follows:

$$
\begin{aligned}
&q = r -^T s = \{t \mid \exists t' \in r, t[A_1,\ldots,A_n] = t'[A_1,\ldots,A_n] \wedge \\
&t[VT_S] = t'[VT_S] \wedge t[VT_E] = t'[VT_E] \wedge t[Per] = t'[Per] \wedge \\
&t[Per_{id}] = t'[Per_{id}] \wedge \neg \exists t' \in r, t[A_1,\ldots,A_n] = t'[A_1,\ldots,A_n] \vee \\
&\exists t' \in r,\ \exists ! t_1,\ldots,t_k \in s,\ \exists ! t'_1,\ldots,t'_s \in \{t_1,\ldots,t_k\}, \\
&\exists f \in fragments(t'[VT_S], t'[VT_E], t_1[VT_S], t_1[VT_E], \ldots, t_k[VT_S], t_k[VT_E]) \\
&t[A_1,\ldots,A_n] = t'[A_1,\ldots,A_n] \wedge t[VT_S] = Start(f) \wedge \\
&t[VT_E] = End(f) \wedge t[Per] = lcm(t'[Per], t'_1[Per], \ldots, t'_s[Per]) \wedge \\
&new_{id} = generate\_id() \wedge t[Per_{id}] = new_{id} \wedge \\
&temp_{id} = generate\_id() \wedge pattern(temp_{id}, t[VT_S], t[Per], \texttt{Periodicity}) = \\
&generate\_union\_pattern(\{t'_1[Per_{id}], \ldots, t'_s[Per_{id}]\}, temp_{id}, t[VT_S], t[Per], \texttt{Periodicity}) \wedge \\
&pattern(new_{id}, t[VT_S], t[Per], \texttt{Periodicity}) = \\
&generate\_difference\_pattern(t'[Per_{id}], temp_{id}, t[VT_S], t[Per], \texttt{Periodicity}) \\
&\wedge pattern(new_{id}, t[VT_S], t[Per], \texttt{Periodicity}) \neq \emptyset\}
\end{aligned}
$$

The function $generate\_difference\_pattern$ (and $generate\_union\_pattern$) is analogous to $generate\_intersection\_pattern$, and generates in a frame time whose duration is the duration of the (new) repetition pattern the new pattern obtained by making the difference of the two input patterns (each ones being a set of periods). $generate\_union\_pattern$ is a slight generalization of such a process, in which union must be performed on an arbitrary number of such patterns. Notice also that the pattern for the union, as generated by the function $generate\_union\_pattern$, is a provisional result, which does not need to be stored into the table `Periodicity`.

The function $fragments$ takes in input the starting (say p_S) and ending (say p_E) point of the frame time to be fragmented, plus an arbitrary number of points (say $p_1,\ldots,p_i$; they are, actually, starting and ending points of other frame times). It provides in output an ordered list of periods, obtained by partitioning the time period $[p_S, p_E]$ into non-overlapping covering parts, having as endpoints p_S, p_E, and all and only those points in $p_1,\ldots,p_i$ which are between p_S and p_E. As a simple example, given the output frame time $t[VT_S] = 50$, $t[VT_E] = 100$ and the frame times $[30,70], [60,80], [50,120], [20,150]$, we have

fragments(50, 100, 30, 70, 60, 80, 50, 120, 20, 150)={[50, 60), [60, 70), [70, 80), [80, 100] }

Given any extended algebraic operator in our approach, and a database of relations expressed in our *implicit* data model, our approach is *correct*, in that it provides *all and only* the results that are provided by applying the corresponding temporal operators (e.g., BCDM or TSQL2 operators) on an *explicit* representation of the same data.

Property 3 (Correcxtness) Our extended algebraic operators, operating on the extended temporal model, are correct.

Notice that, as discussed in section 3, for the sake of generality we may also represent non-periodic valid-time tuples in our model, using periodic tuples having NULL values for the *Per* and Per_{id} attributes. The above definitions of extended temporal relational operators can be easily extended to cope with them. For instance, let us focus on Cartesian Product (the treatment of the other operators is similar). Trivially, we can say that, in case both tuples are non-periodic, only the part of the definition concerning non-temporal attributes and the frame times (modelling the valid times of the tuples) has to be considered. In such a case, the resulting operator simply performs the concatenation of non-temporal parts, and the intersection of valid times, as demanded by the BCDM model and by TSQL2. Analogously, when a periodic tuple and a non-periodic ones have to be combined through Cartesian Product, the resulting tuple has frame time the intersection of the frame times, and as periodic pattern (and duration of the periodic pattern) the one of the periodic tuple[6].

An analogous generalization is trivially possible also for the other temporal extensions of the other relational algebra operators. In this sense, we can say that the following property holds:

Property 4 (consistent extension) Our temporal relational algebra is a "consistent extension" of the BCDM (and TSQL2) algebra.

5 Empirical testing

In order to show the practical relevance of our implicit approach to efficiently manage periodic data, we have performed an extensive experimental evaluation. In particular, we have compared the performance of our approach with respect to the one of the standard explicit one.

[6] For instance, the above definition of Cartesian Product can be extended to include the following cases:
$q = r \times^T s = \{t | \ldots \vee$
$\exists t_r \in r, \exists t_s \in s,\ t_r[Per] = NULL \wedge t_s[Per] = NULL \wedge$
$t[A_1,\ldots,A_n] = t_r[A_1,\ldots,A_n] \wedge t[B_1,\ldots,B_k] = t_s[B_1,\ldots,B_k] \wedge t[VT_S] = max(t_r[VT_S],t_s[VT_S]) \wedge t[VT_E] = min(t_r[VT_E],t_s[VT_E]) \wedge$
$t[Per] = NULL \wedge t[Per_{id}] = NULL \wedge t[VT_S] < t[VT_E] \vee$
$t_r[Per] \neq NULL \wedge t_s[Per] = NULL \wedge t[A_1,\ldots,A_n] = t_r[A_1,\ldots,A_n] \wedge t[B_1,\ldots,B_k] = t_s[B_1,\ldots,B_k] \wedge$
$t[VT_S] = max(t_r[VT_S],t_s[VT_S]) \wedge t[VT_E] = min(t_r[VT_E],t_s[VT_E]) \wedge t[VT_S] < t[VT_E] \wedge t[Per] = t_r[Per] \wedge$
$t[Per_{id}] = t_r[Per_{id}] \vee$
$t_r[Per] = NULL \wedge t_s[Per] \neq NULL \wedge t[A_1,\ldots,A_n] = t_r[A_1,\ldots,A_n] \wedge t[B_1,\ldots,B_k] = t_s[B_1,\ldots,B_k] \wedge$
$t[VT_S] = max(t_r[VT_S],t_s[VT_S]) \wedge t[VT_E] = min(t_r[VT_E],t_s[VT_E]) \wedge t[VT_S] < t[VT_E] \wedge t[Per] = t_s[Per] \wedge$
$t[Per_{id}] = t_s[Per_{id}]\}$

We remark here that, with the term "explicit" approach, we mean the approach in which periodic data are explicitly stored (see e.g., table Activity_Expl in Section 3), so that queries operate directly on such a representation.

In the experiments, we have adopted the following methodology to index data. Since it has been show in the literature that the RI-Tree Kriegel et al (2000) has the best performance considering the Physical disk I/O and the query response time and at the same time can be employed within commercial RDBMS, we decided to employ the RI-tree in our implementation. Specifically, we index the VT_S and VT_E temporal attributes of periodic relations using the RI-tree. For the sake of fairness, also the data in the *explicit* approached have been indexed in the same way.

All experimental results presented in this section are computed on a four 450MHZ CPU - SUN UltraSparc II processor machine, running Oracle 10.2.0 RDBMS, with a database block size of 8K and SGA size of 100MB. At the times of testing the database server did not have any other significant load. We used Oracle built-in methods for statistics collection, analytic SQL functions, and the PL/SQL procedural runtime environment.

We chose to compare our results considering the following parameters: space usage, physical I/O, CPU usage, and query response time. We will especially focus on physical I/O, since it is usually considered to be the most important one while evaluating efficiency of accessing data Hellerstein et al (1997).

5.1 Data sets

Our approach is domain and application independent. For the testing, we have taken advantage of the previous experience of some of the co-authors in the medical informatics domain.

As a matter of fact, many activities are routinely executed at periodic time by nurses on hospitalized patients. Additionally, many medical therapies are a significant example of activities to be repeated at a periodic time Anselma et al (2006). There are also cases of *open-ended* repeated activities (e.g., dialysis on diabetic patients must usually be performed twice or three times each week, for all the life of the patient). Moreover, data in hospitals are necessarily historical, since hospitals need to maintain the past history of their patients (as well as to store future data to schedule part of patients' future treatments).

The previous research activity of some of the authors concerned "prototypical" medical data, since privacy motivations impose that we do not have currently available real data. In absence of real data, based on our experience, we have generated periodic data to simulate real applications scenarios. The following data distributions parameters have been considered (we used hour as the basic granularity):

- Number of Patients **16,824**
- Average number of periodic activities per patient **8.30**
- Average number of periods in a periodic pattern = **4.86**
- Average duration of period of periodic patterns = **87.56**
- Average duration of the frame time **1169**

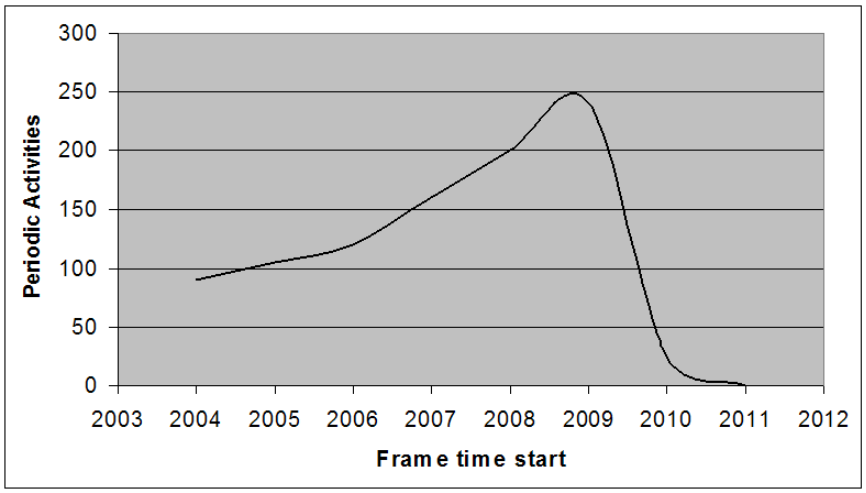

Fig. 1 Average number of new periodic activities per day over the Frame time start

- Distribution of the frame time – see Figure 1 where we assume NOW=January 1, 2009; It can be seen that during the past the number of activities is slightly increasing, while future data are reducing (in fact, it is likely to know activities in next week or month but not much more far in the future).
- Distribution of the duration of periodic pattern; this parameter is shown in Table 4, where it can be seen that the majority of periodic activities are repeated daily or weekly.
- Non periodic tuples constitute about 5% of the data.

The above mentioned parameters represent a real world scenario in a small hospital, which has relatively small number of patients, and of activities, and have been chosen on the basis of an evaluation of real periodic data in hospital. Despite periodic data in a hospital include also open ended data (since the ending time of the frame time may be unknown, as in the case of patients needing a therapy for all the rest of their life) in this testing we could not include such data because the explicit representation cannot support it.

5.2 Experiment: data structures

In order to carry on the experiments, the same periodic activities concerning hospital patients have been represented both in the *implicit* and *explicit* model. In the implicit model, the representation of data required 353,367 records in the `Activity` table and about 2 million records in the `Periodicity` table. In order to represent the same activities in the explicit model, more than 194 million records are required in the `Activity_Expl` table, as shown in Tables 5 and 6.

As expected, the adoption of an implicit representation provides clear advantages as regards storage. For our medical real world scenario, the explicit method requires

Duration Hour	*Pecentage %*
1	2.00
2	1.57
4	2.07
6	5.08
12	9.29
24	37.94
48	6.09
168	31.12
720	2.53
Random	2.30

Table 4 Distribution of duration of period of repetition.

Table Name	*Number of records*	*Table size MBytes*	*Primary Index MBytes*	*Upper Index MBytes*	*Lower Index MBytes*	*Total MBytes*
Activity	353,367	16.25	8.00	8.00	5.19	37.44
Periodicity	2,108,495	43.08	37.00	0	0	80.08
TOTAL						117.52

Table 5 Space requirements for the Implicit model data

Table Name	*Number of records*	*Table size MBytes*	*Primary Index MBytes*	*Upper Index MBytes*	*Lower Index MBytes*	*Total MBytes*
Activity_Expl	194,671,463	7,331.82	3,520.00	4,288.00	4,288.00	19,427.83
TOTAL						

Table 6 Space requirements for the Explicit model

more than 160 times more storage space for efficient management comparing to our implicit method, as it can be seen in Tables 5 and 6.

As discussed above, we used RI-tree for indexing for both `Activity` and `Activity_Expl` tables.The RI-tree method requires that initial tables are altered with column *node*, which is calculated for every row of data by algorithm Kriegel et al (2001). Also, two B^+-tree composite indexes have been created *LowerIndex* (node, VT_S) and *UpperIndex* (node, VT_E). Range queries are performed by calling the dedicated procedure that collects leftnodes and rightnodes into temporary tables and then performs the transformed SQL statement as instructed in Kriegel et al (2000).

To ensure that we can collect accurate information about physical disk reads on data and index structures and also CPU usage, we used Oracle built-in methods for statistics collection and queried V$FILESTAT, V$DATAFILE and V$SYSTAT views. Space usage for Tables and Index structures are collected from the data dictionary view User_Segments.

5.3 Temporal algebraic operators: Analysis and Results

For the sake of brevity, in this paper we focus only on temporal Cartesian Product, which is the basic operator in order to ”join” two relations. Given the nature of Cartesian Product, which pairwise combines all the tuples of the two input tables, directly running Cartesian Product on the whole dataset was not possible. We had to to generate smaller datasets of reasonable sizes.

In our approach, Cartesian Product (as well as the other algebraic operators) applies to an implicit representation, and provides as output an implicit representation. In several cases, an ”implicit” output is more commonsense and desirable than an explicit one. Since it is smaller of the corresponding explicit one, our implicit approach clearly outperforms the explicit one.

In order to make the comparison with the explicit approach more complete and fair, in the experiments we also took into account a variant for our approach. We considered the case in which, after the execution of our temporal Cartesian Product, the implicit output is made explicit through an application of the *make_explicit* function. Considering such a variant (indicated by ”IMP+MakeEXP” in the rest of this section), the output size is exactly the same as the one of the explicit approach.

For the sake of brevity, in this paper we report results concerning the response time only. As a matter of facts, CPU time behaves in the same way, because disk I/O does not play a significant role in this experiment, given the relatively small input size.

In figure 2 we compare our approach (”IMP”) and its variant (”IMP+MakeEXP”) with the explicit one (”EXP”) considering different input sizes (the input size we show in the figure is the one of the ”explicit” approach).

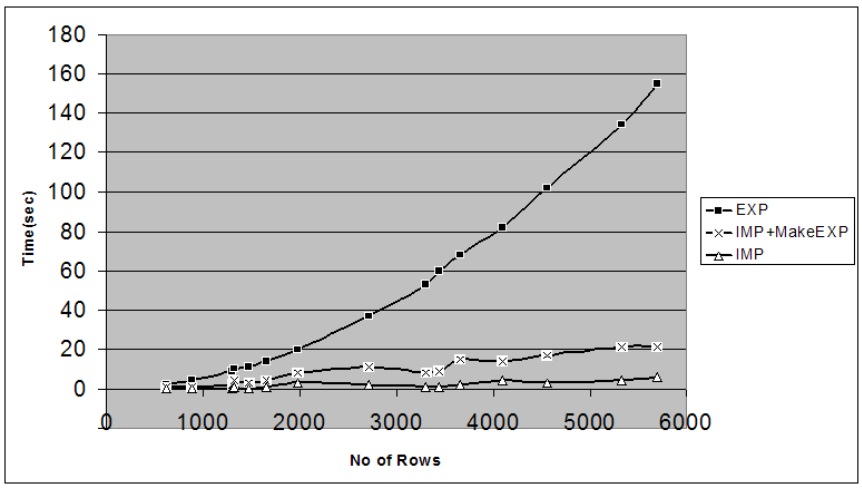

Fig. 2 Temporal Cartesian Product Response Time

The experiment clearly shows the advantages of our implicit approach, and even of its variant. Specifically, while the increase of the input size slightly affect our approach, the response time of the explicit approach significantly increases with it. For instance, with input size 900 rows, IMP response time is less than 1 second, IMP+MakeEXP is 1 second, while EXP is 4 seconds; With input size 2700 rows, IMP is 2 seconds, IMP+MakeEXP is 11 seconds, while EXP is 37 seconds; With input size 5700 rows, IMP is 6 seconds, IMP+MakeEXP is 21 seconds, while EXP is 155 seconds.

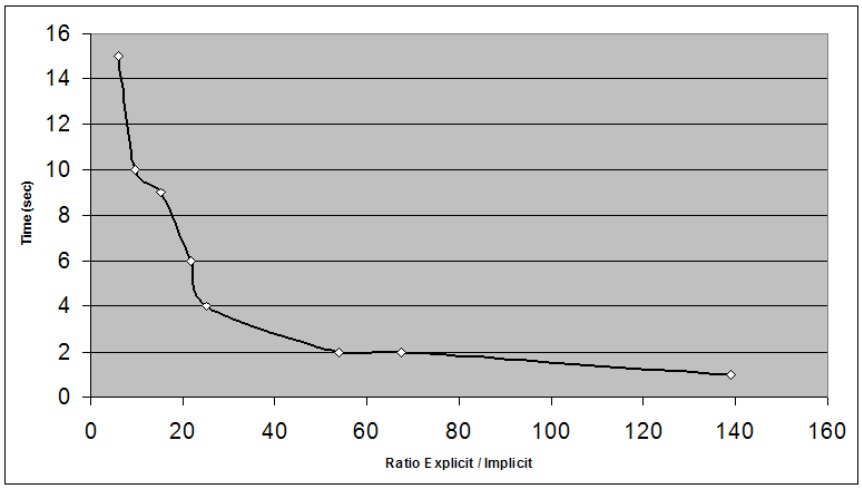

Fig. 3 Implicit Temporal Cartesian Product Response Time as a factor of different Explicit/Implicit ratios

In figure 3, we present the response time of our (IMP) approach considering different values for the explicit/implicit i.e., the ratio between the size of the explicit representation and the size of corresponding implict one, and considering an approximatively fixed input size (about 3700 tuples) and output size for the corresponding explicit approach. Notice that the explicit approach is not affected at all by the ratio, since the answer size is constant, and its response time is 68 seconds for all the experiments. On the other hand, considering the IMP+MakeEXP variant, the *make_explicit* function adds a constant overhead of about 10 seconds to the IMP approach. It is worth noticing that, as expected, the greatest is the ratio, the greatest is the gain of our implicit approach. For example, when the ratio is 5, the response time is 15, when the ratio is 15, the response time is 9, and when the ratio is 50, the response time is 2 (i.e., less than 1/30 of the explicit approach).

5.4 Other issues

In the above experiments, we have compared the performance of our approach with respect to the traditional explicit one, showing its advantages. Finally, however, it is

worth stressing that there are also additional advantages, in that the implicit approach we propose can deal with aspects that cannot be coped with by the explicit one. Open-ended periodic data cannot even be represented in the explicit approach. Notice that such data are relevant in many application domains, including the medical one we have discussed in this paper (in the medical domain, chronical diseases such as diabetes require treatments to be carried on for all the life of patients). Last, but not least, it is worth stressing that, using an explicit approach to periodic data in domains such as manufacturing, in which the "explicit/implicit" ratio may be very high, storage is likely to be saturated, so that only a limited temporal window of temporal data can be maintained, and suitable vacuuming strategies need to be devised.

6 Conclusion, Comparisons and Future Work

In this paper, we propose a new approach to cope with periodic data in relational databases. Specifically:

- we have proposed an "implicit" relational data model for user-defined periodic data, which is based on the "consensus" definition of granularity in the temporal database glossary Bettini et al (1998) and its extension to cover periodic granularities in Bettini and De Sibi (2000), and is a "consistent extension" of TSQL2's one Snodgrass (1995).
- we have extended Codd's algebraic operators of Cartesian product, Union, Projection, nontemporal selection, and Difference, in order to provide a complete query language coping with implicit periodic data; Such operators are correct, and are a consistent extension of BCDM (and TSQL2) algebra;
- finally, we have developed an extensive experimentation of our model and methodology, showing that our "implicit" approach overcomes the performance of traditional "explicit" approaches.

As already mentioned in the introduction, there are several 'implicit' approaches to user-defined periodic data in both the Artificial Intelligence and Database area, not to mention Temporal logics (see, e.g., the survey in Tuzhilin and Clifford (1995)). However, as already mentioned, non of themsatisfied all the (1)-(4) desiderata we have definined in the introducion.

In the following, we just focus on the most closely related ones. In the best of our kwnoledge, the only implicit representation formalism based on the "consensus" definition of periodic granularities has been proposed also by Ning et al (2002). However, they mostly focused on the representation formalism to deal with periodic granularities, and on the definition of the language operators needed in order to define new granularities on the basis of other granularities. On the other hand, they did not focus on a representation of such granularities in the relational model, and on the related issues of extending the relational algebra to query such data. To the best of our knowledge, the only approaches focusing on an extension of algebraic operations to cope with implicit periodic relational data are Terenziani (2003), Kabanza et al (1995), and Niezette and Stevenne (1992). None of these approaches, however, copes with the definiotion of periodic and quais-periodic granularity in the glossary and, in

particular, none of them copes with exceptions. Additionally the properties of being (both data model and algebra) a consistent extension of TSQL2 does not hold for such approaches. Terenziani has proposed a representation formalism to cope with periodic data in an implicit way, which was mainly an adaptation of Leban's one Leban et al (1986). He also provided an extended temporal algebra operating on the new formalism, as well as symbolic and semi-symbolic algorithms to implement the algebraic operations on the new data model. However, the complexity of the extended algebraic operations is shown to be exponential, since the number of "terms" defining in a implicit way the valid time of tuple grows exponentially in the number of algebraic operations being performed Terenziani (2003). Kabanza et al (1995) have defined a *constraint-based* formalism based on the concept of linear repeating points (henceforth lpr's). A lrp is a set of points $\{x(n)\}$ defined by an expression of the form $x(n) = c + kn$ where k and c are integer constants and n ranges over the integers. A generalized tuple of temporal arity k is a tuple with k temporal attributes, each one represented by a lrp, possibly including constraints. For instance, the generalized tuple $(a_1,\ldots,a_n|[5+4n1,7+4n2] \wedge X_1 = X_2 - 2)$ (with data part $a_1,\ldots,a_n$) represents the infinite set of tuples $\{(a_1,\ldots,a_n|[1,3]),(a_1,\ldots,a_n|[5,7]),(a_1,\ldots,a_n|[9,11]),\ldots\}$ or, in other words, a tuple $(a_1,\ldots,a_n)$ having an infinite periodic valid time. A generalized relation is a finite set of generalized tuples of the same schema. In Kabanza et al (1995), the algebraic operations have been defined over generalized relations as mathematical manipulations of the formulae coding lrp's, and the complexity of such operations has been proven to be exponential. Moreover, the expressiveness of the proposed formalism has been analysed, in terms of Presburger's Arithmetics. Niezette and Stevenne (1992) have proposed a symbolic extension to Kabanza's approach, mostly providing a symbolic formalism as an interface language, whose meaning is defined in terms of the underlying lrp expressions.

In our approach, we have overcome such a general limitation of implicit relational approaches to periodic temporal data. Additionally, the extensive experimental comparison between our implicit approach an the "traditional" explicit one is one of the core contributions of our work, showing the computational advantage of our implicit approach with respect to the explicit one.

In this paper, we have focused on the internal representation of (quasi)-periodic granularities. On top of such a representation, high-level algebraic (also called symbolic) languages, such as the ones in Bettini and De Sibi (2000), Egidi and Terenziani (2005), Leban et al (1986), Niezette and Stevenne (1992), Ning et al (2002), Terenziani (2003) could be added, in order to provide a high-level, user-friendly and (hopefully) commonsense interface to help users in defining user-defined (quasi)-periodic granularities in a user-friendly and compositional way. The construction of such an additional layer is outside our current goals, and will be addressed as future work.

References

Anselma L, Terenziani P, Montani S, Bottrighi A (2006) Towards a comprehensive treatment of repetitions, periodicity and temporal constraints in clinical guidelines. Arificial Intelligence in Medicine 38(2):171–195

Bettini C, De Sibi R (2000) Symbolic representation of user-defined time granularities. Annals of Mathematics and Artificial Intelligence 30(1-4):53–92, URL citeseer.ist.psu.edu/bettini99symbolic.html

Bettini C, Dyreson C, Evans W, Snodgrass R, Wang X (1998) A glossary of time granularity concepts. In O Etzion, S Jajodia, and S Sripada, editors, Temporal Databases: Research and Practice, number 1399 in LNCS State-of-the-art Survey, Springer-Verlag pp 406–413

Bettini C, Jajodia S, Wang S (2000) Time Granularities in Databases, Data Mining, and Temporal Reasoning. Springer Verlag

Böhlen MH, Snodgrass R, Soo M (1996) Coalescing in temporal databases. In: Procs. VLDB, pp 180–191

Chomicki J, Imielinski T (1993) Finite representation of infinite query answers. ACM ToDS 18(2):181–223

Codd E (1971) Relational completeness of data base sublanguages. In: Courant Computer Science Symposia 6, Data Base Systems, pp 24–25

Dyreson CE, Snodgrass RT, Freiman M (1995) Efficiently Supporting Temporal Granularities in a DBMS. Tech. Rep. TR 95/07, URL citeseer.nj.nec.com/dyreson95efficiently.html

Egidi L, Terenziani P (2004) A lattice of classes of user-defined symbolic periodicities. In: Proc. 11th International Symposium on Temporal Representation and Reasoning, pp 21–27

Egidi L, Terenziani P (2005) A flexible approach to user-defined symbolic granularities in temporal databases. In: Procs. of ACM SAC'05, pp 592–597

Egidi L, Terenziani P (2006) A mathematical framework for the semantics of symbolic languages representing periodic time. Annals of Mathematics and Artificial Intelligence 46(3):317–347

Egidi L, Terenziani P (2006) A mathematical framework for the semantics of symbolic languages representing periodic time. Annals of Mathematics and Artificial Intelligence 46(3):317–347

Egidi L, Terenziani P (2008) A modular approach to user-defined simbolic periodicities. Data & Knowledge Engineering 66(1):163–198

Gao D, Jensen C, Snodgrass RT, Soo M (2005) Join Operations in Temporal Databases. VLDBJ 14:2–29

Hellerstein J, Koutsupias E, Papadimitriou C (1997) On the Analysis of Indexing Schemes. 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems

Jensen CS, et al (1998) The consensus glossary of temporal database concepts, in temporal databases: Research and practice, o. etzion, s. jajodia, and s. sripada (eds). Springer-Verlag pp 367–405

Jensen CS, Snodgrass RT (1996) Semantics of Time-Varying Information. Information Systems 21(4):311–352

Kabanza F, Stevenne JM, Wolper P (1995) Handling infinite temporal data. Journal of Computer and System Sciences 51:3–17

Kriegel HP, Ptke M, Seidl T (2000) Managing intervals efficiently in object-relational databases. Proceedings of the 26th International Conference on Very Large Databases pp 407–418

Kriegel HP, Potke M, Seidl T (2001) Object-relational indexing for general interval relationships. In: Proc. 7th Intl Symposium on Spatial and Temporal Databases (SSTD01)

L Edwin McKenzie J, Snodgrass RT (1991) Evaluation of Relational Algebras Incorporating the Time Dimension in Databases. ACM Computing Surveys (CSUR) 23(4):501–543

Leban B, McDonald D, Forster D (1986) A representation for collections of temporal intervals. In: Procs. of AAAI'86, pp 367–371

Liu L, Tamer zsu M (2009.) Encyclopedia of Database Systems. Springer Verlag

Niezette M, Stevenne JM (1992) An efficient symbolic representation of periodic time. In: Procs. of CIKM

Ning P, Wang X, Jajodia S (2002) An algebraic representation of calendars. Annals of Mathematics and Artificial Intelligence 36(1-2):5–38

Snodgrass RT (1995) The TSQL2 Temporal Query Language. Kluwer Academic

Snodgrass RT, Ahn I (1985) A Taxonomy of Time in Databases. In: Navathe SB (ed) Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data, ACM Press, pp 236–246

Soo M, Snodgrass R (1993) Multiple calendar support for conventional database management systems. In: Proc. Int'l Workshop on an Infrastructure for Temporal Databases

Terenziani P (2000) Integrated temporal reasoning with periodic events. Computational Intelligence 16(2):210–256

Terenziani P (2003) Symbolic user-defined periodicity in temporal relational databases. IEEE TKDE 15(2):489–509

Terenziani P (2009) Temporal Periodicity, Springer Verlag, Ling Liu and M. Tamer zsu Editors, pp 3004–3008

Tuzhilin A, Clifford J (1995) On periodicity in temporal databases. Information Systems 20(8):619–639