An implicit approach to deal with periodically-repeated medical data

Bela Stantic[a], Paolo Terenziani[b,*], Guido Governatori[c], Alessio Bottrighi[b], Abdul Sattar[a]

[a]*Griffith University, Brisbane, Australia*
[b]*Universita' del Piemonte Orientale, Alessandria, Italy*
[c]*National ICT Australia, Brisbane, Australia*

Abstract

Temporal information plays a crucial role in medicine, so that in medical informatics there is an increasing awareness that suitable database approaches are needed to store and support it. Specifically, a great amount of clinical data (e.g., therapeutic data) are periodically repeated. Although an explicit treatment is possible in most cases, it causes severe storage and disk I/O problems. In this paper, we propose an innovative approach to cope with periodic relational medical data in an implicit way. We propose a new data model, representing periodic data in a compact (implicit) way, which is a consistent extension of TSQL2 consensus approach. Then, we identify some important types of temporal queries, and present query answering algorithms to answer them. Finally, we show experimentally that our approach outperforms current explicit approaches.

Keywords: Periodic Temporal Data, Temporal Databases, Data Model, Experimental Evaluation

1. Introduction

Most clinical data (e.g., patients' clinical records) are naturally temporal. In order to be meaningfully interpreted, patients' symptoms, laboratory test results, and, in general, all clinical data, must be paired with the time in which they hold (called *valid time* henceforth). In many cases, medical data concerns events that have to be repeated at periodic time. Such events include, e.g., routine activities that nurses have to perform daily on hospitalized patients, as well as intrinsically repeated activities such as chemotherapy cycles, or dialysis (which is usually an open-ended activity, since it has to be performed for all the life of certain diabetic patients). An explicit representation of all the repetitions to be performed might be important, e.g., for scheduling purposes and resource allocation. Nevertheless, it is very costly, both in terms of storage allocation, and of disk I/O when data have to be retrieved.

1.1. Periodic data in databases

Unfortunately, the research about temporal data has widely demonstrated that the simple addition of some timestamped attributes (e.g., the START and END times for the valid time of a tuple) is not enough, since many complex problems need to be tackled. *"<<Two decades of research into temporal databases have unequivocally shown that a time-varying table, containing certain kinds of DATE columns, is a completely different animal than its cousin, the table without such columns.*

*Corresponding author: tel: +390131360174 fax: +390131360198
E-mail address: terenz@mfn.unipmn.it (P. Terenziazi).

Effectively designing, querying, and modifying time-varying tables requires a different set of approaches and techniques than the traditional ones taught in database courses and training seminars. Developers are naturally unaware of these research results (and researchers are often clueless as to the realities of real-world application development). As such, developers often reinvent concepts and techniques with little knowledge of the elegant conceptual framework that has evolved and recently consolidated... >>" in [1], Section "Preface", Subsection: "A paradigm shift", page XVIII.

For instance, Das and Musen have identified several types of mismatches between the temporal support of standard databases and the richness of clinical data [2]; analogously, James and Goble [3] have pointed out the requirements that medical records impose on a temporal model. Designing, querying and modifying time-varying tables requires a different set of techniques. Such techniques have been studied in more than 25 years of research by the temporal database (TDB henceforth) community (consider, e.g., the overview [4]). Although TDB is still an open area of research, many researchers have already consolidated a "basic core" of results, by defining the TSQL2 consensus approach [5]. In the medical area, several temporal database approaches have been devised. For instance, Chronus [6] and Chronus II [7] have provided an implementation of a subset of TSQL2 [5], with specific focus on valid time. Terenziani and others have explored the impact of the telic/atelic dichotomy [8] on medical data [9].

On the other hand, although periodic data[1] are quite frequent in the medical context, no approach has been developed in order to cope with such data in an efficient way. For instance, since periodic actions are an intrinsic constituent of clinical guidelines, several approaches in the area have devised expressive languages to represent complex periodic patterns (such as, e.g., those in chemotherapy treatments). Among others, Asbru's [10] and GLARE's [11] [12] temporal languages have been devised to model complex cases of periodically repeated actions. However, while in Asbru's and GLARE's languages repetition patterns in the guidelines can be represented, to the best of our knowledge no medical database approach has been devised to store in a (relational) database the actual data modelling the effective execution of repeated actions (e.g., dialysis) on each specific patients on which it has to be physically executed.

1.2. Explicit vs. implicit approaches

The trivial way to store a repeated action in a database is to explicitly store all the repetitions of that action. E.g., consider the following therapy for multiple mieloma (such a therapy has been used as one of the example of application of GLARE's temporal representation language [11]).

Example 1 The therapy for multiple mieloma is made by six cycles of 5-day treatment, each one followed by a delay of 23 days (for a total time of 24 weeks). Within each cycle of 5 days, 2 inner cycles can be distinguished: the melphalan treatment, to be provided twice a day, for each of the 5 days, and the prednisone treat-

[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.

ment, to be provided once a day, for each of the 5 days. These two treatments must be performed in parallel.

It may be important that such periodic actions (data) are recorded in some way. For instance, such data are important in order to schedule hospital personnel activities, and to manage resource allocation.

While GLARE's representation language provides an high-level language to represent such a periodic pattern, a separate problem is to provide a proper support to store the time of execution the actions on specific patients affected by multiple mieloma.

There is an obvious and trivial way to cope with 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 a standard relational database approach it would consist, for each patient, of at least 90 tuples, modelling 60 melphalan applications, and 30 prednisone applications. 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, the "explicit" approach has a main disadvantages with respect to the implicit one:

- it is very expensive in term of physical disk I/O's, due to the high storage size. In many practical application area, the number of repetitions (at periodic time) of activities is high, and thus making explicit all the repetitions is space-demanding. Making all such data explicit might rapidly reach a critical data size even for the most efficient commercial DBMS, with dramatic consequences especially in terms of physical disk I/O's (see the experiments in Section 5, considering the medical domain).

Additionally:

- the "explicit" approach 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 all 90 actions is not probably the most user-friendly and perspicuous answer to a user wanting to know when activities must be performed;
- the "explicit" approach 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; consider, e.g. dialysis). 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.

However, an implicit treatment of periodic data involves a major departure from traditional relational database techniques, and a switch towards the Artificial Intelligence techniques. Indeed, a common underlying assumption of databases (with the exception of inductive and logical databases) is that the data they contain are explicit: tuples explicitly model all and only the facts that one wants to consider. In such a context, there is no need of any inferential mechanism to derive new conclusions from the basic data: all conclusions are already explicitly stated in the database. However, if we move towards an implicit treatment of periodic data, the above underlying assumption is no more valid: not all data are explicit, and there must be some technique to 'make implicit knowledge explicit' when required. As we will further discuss in section 3.1 in the

following, symbolic manipulation techniques are thus required, thus leading to the development of Artificial Intelligence techniques.

For such reasons, in the area of temporal databases, only few initial approaches have been devised to provide an implicit representation of periodically repeated data (consider, e.g., [13], [14], [15]; see Section 6). In such approaches, periodically repeated data are not explicitly elicited: on the other hand, the pattern of repetition is directly stored in the database, so that a compact representation is achieved. However, to the best of our knowledge, no "implicit" approach to periodic data in the literature has explicitly focused on issues related to the efficient representation and management of periodical data. In this paper, we describe an approach overcoming such a limitation, with specific focus on medical data.

1.3. Goals and methodology

In summary, although there seems to be a general agreement within the Database and 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

- a relational implicit representation
- additional temporal operators, to ask, e.g., range queries
- indexing and access

However, all such issues are fundamental for the practical applicability of any medical Database considering periodic data. In our approach, we merge relational database and Artificial Intelligence techniques in order to devise such a comprehensive approach and in a "principled way". Specifically, our approach has been designed in such a way that

(1) our data model has the expressiveness to capture all "periodic granularities", as defined in the Database literature [16], [17] (see property 1 defined further on in the paper),

(2) our data model is a consistent extension of the TSQL2 consensus model [5] (see property 2 defined further on in the paper), and

(3) our temporal query operators are correct with respect to conventional explicit approaches, in which all the repetitions of periodic data are explicitly stored (see property 3 defined further on in the paper).

Property 1 grants that the expressiveness of our data model is the one requested by the temporal Database literature. On the other hand, property 2 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 2 is essential, since it grants the interoperability of our approach with pre-existent TSQL2 and with standard relational data. Finally, property 3 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 and query response time.

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, or deleted from' the database [18]) since no periodicity issue is usually involved by it. 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). As a consequence, the approach dealing with the (periodic) valid time proposed in this paper can be trivially extended to deal 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., [5], [15], [19], [20]).

1.4. Summary

The rest of the paper is organized as follows. Section 2 is a preliminary one, in which we first briefly recall TSQL2 approach, and then we report the basic definitions of periodic and quasi-periodic temporal granularities [16] (where quasi-periodic granularities extend periodic granularities to cope with *finite* exceptions). In Section 3 we first propose an abstract implicit representation of quasi-periodic granularities, and then we propose an extended relational temporal data model coping with it. Temporal *range queries* are particularly important in the temporal Database context [21]. In Section 4 we identify different types of temporal range queries in the context of periodic data, and we devise algorithms to cope with them, showing that they are correct. In section 5, we present an experimental evaluation of our approach, showing its advantages with respect to the "traditional" explicit approach. In section 6, we present related works and comparisons. Finally, section 7 addresses conclusions, comparisons and future work.

A preliminary and short version (5 pages) of the work reported in this paper has been published in the Proceedings of the Medinfo'2010 Conference [22].

2. Preliminaries

In order to set the stage, we first briefly introduce TSQL2. We then introduce the basic definitions in the Database literature on which our approach is grounded.

2.1. 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 1 (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 an "implicit" way, with "quasi-periodic" data.

Definition 2 (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).

2.2. (Quasi)-Periodic granularities

In this preliminary section, we give the definition of granularity taken from the temporal database glossary [17], and its successive extension to cover periodic and quasi-periodic temporal granularities [16]. 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 3 (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 $(\mathbb{Z}, \leq)$, $(\mathbb{N}, \leq)$, or $(\mathbb{R}, \leq)$.

Definition 4 (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 5 (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 6 (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. For the sake of clarity, we will adopt Example 1 to exemplify them, but we stress that our methodology is general. In particular, we focus on the administration of prednisone to a specific patient, starting, e.g., at DAY 100 (we call PRED such a granularity).

Definition 7 (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 $H(i+n) = \bigcup_{r=0}^{k} G(j_r+m)$.

In the definition, each granule $H(i)$ of H is constituted by $k+1$ granules of G, n is the cardinality of the repetition pattern (i.e., the number of periods it contains), and m is its duration. For instance, let us consider the user-defined granularity PRED in Example 1. Each granule in PRED is composed by a set of granules of DAY (i.e., DAY *groups into* PRED). Indeed, this is a specific case with respect to the general definition 7, since each granule in PRED is composed by exactly one granule of DAY (i.e., $k = 0$ in the above definition). For instance, the first granule PRED(0) of PRED is the granule DAY(100). The second condition in definition 7 demands that such a "group into" relation must be characterized by a periodic repetition of the "grouping pattern". An instance of the "grouping patters" (e.g., the instance concerning PRED(0)) is:

$\text{PRED}(0) = \{DAY(100)\}$
$\text{PRED}(1) = \{DAY(101)\}$
$\text{PRED}(2) = \{DAY(102)\}$
$\text{PRED}(3) = \{DAY(103)\}$
$\text{PRED}(4) = \{DAY(104)\}$

This pattern repeats every 28 granules of DAY (e.g., each four weeks). Therefore, considering definition 7, for example 1 we have n=5 (since the periodic pattern contains 5 periods) and m=28, since its duration is four weeks (i.e., 28 days). For instance, from the definition 7 above, we trivially have that $(\text{PRED}(0+5) = \{\text{DAY}(100+28)\}$.

Intuitively, the quasi-periodic groups-into relation is basically a periodic *groups-into* relation, but with the addition of some "additional granules", which are not periodic. For example, let us suppose that two additional administrations of prednisone have to be performed on the patient at days 268 and 269 (henceforth we call PRED+ such a granularity).

Definition 8 (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 $H(i+n) = \bigcup_{r=0}^{k} G(j_r+m)$.

$E_1, \ldots, E_z$ represent the additional granularities to be added to the periodic pattern (i.e., the positive exceptions). Intuitively, with respect to definition 7 above, this definition restricts the periodical behavior of the granules of H, characterized by n and m, to hold only between pairs of consecutive exceptions. I.e., it states that the relation $H(i+n) = \bigcup_{r=0}^{k} G(j_r+m)$ holds only for those granules such that there are no exceptions between them. This is trivially true in our example (again, with $n = 5$ and $m = 28$), where the exceptions occur after all the granules of PRED+ that repeat in a periodic way. Thus, from definition 8 we have that DAY *quasi-periodically groups into* PRED+.

Finally, the definition of periodic and quasi-periodic granularities is given in [16] in terms of a bottom granularity.

Definition 9 (Quasi-periodic granularity) A periodic (resp. quasi-periodic) granularity is a granularity periodic (resp. quasi-periodic) with respect to the bottom granularity.

As a consequence, taking days (or hours, or minuts, etc.) as the bottom granularity, we have that that (from definition 9) PRED is a periodic granularity and PRED+ is a *quasi-periodic granularity*.

3. Representing (quasi)-periodic granularities

In this section, we propose a representation of (quasi)-periodic granularities, based on the above general definitions.

3.1. Symbolic manipulation on databases

Before moving towards the representation, it is worth considering the implications of the move towards an implicit representation of periodic data.

In Figure 1, we graphically show the basic notions underlying our representation. The "Explicit Representation" part of the figure shows what periodic data actually is: a sequence of time periods which repeat regularly on the time line. Usually, only a part (henceforth called Frame Time) of the whole timeline is considered (i.e., repetitions are bounded). Given some (bounded or unbounded) periodic activity, a *range* query may ask whether such an activity has to be performed within a specific time period Q or not (Q has been delimited by dotted vertical bars in Figure 1). If we have in the database an explicit representation of all the time periods for the periodic activity, the range query can be easily answered by looking whether there is an intersection between those periods and Q (see the upper part of Figure 1).

However, as discussed in the introduction, an *explicit* representation of periodic data has several limitations. An *implicit* representation can be used to overcome them. In an implicit definition, one can isolate the pattern of periods that repeat regularly in time, and the duration 'P' at which they repeat. Only one pattern must be explicitly represented, with the intended meaning that such a pattern is a sort of "prototype" that repeats regularly every 'P' in time. In other words, an implicit representation is a compact representation, whose explicit meaning is the corresponding explicit model. The implicit representation is space-effective, but answering the above range query Q on the basis of the implicit representation is more complex with respect to the conventional explicit case, since data are not explicitly available. Indeed, a trivial strategy could be to convert all implicit data into explicit ones before answering queries. However, such a strategy is inefficient, both from the space and the time complexity points of view. A more efficient strategy requires that query answering operates directly on the implicit representation, performing some form of symbolic manipulation on it. In such a way, one can operate on the compact representation, saving space and, hopefully, time. This strategy involves the development of a typical Artificial Intelligence approach, operating symbolic manipulation and inferences on implicit (and compact) representations of data. Also, it requires a form of theoretical validation: we to prove that the results we obtain by operating symbolically on the implicit representation are the same that would be obtained (in a much less efficient way) by operating on explicit data (as in the usual database ap-

proaches). Such a correctness property has been proved as regards our symbolic approach to query answering.

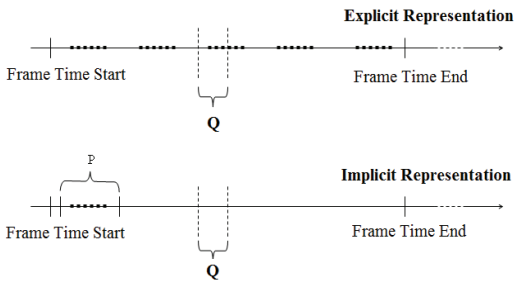

Figure 1: Representations of Periodic data

We first provide a suitable "abstract" representation of quasi-periodic granularities (subsection 3.2), and then we identify its counterpart in a relational environment (subsection 3.3).

3.2. An implicit representation of quasi-periodic granularities

In order to provide an 'abstract" implicit representation of quasi-periodic granularities, we start from the below property, taken from [16]:

Property 1 (finite representation)

The *groups periodically into* relation guarantees that granularity H can be finitely described in terms of granules of G providing the following information:

(i) the finite sets $S_0, \ldots, S_{n-1}$ of indexes of G each one describing the composition of one of the n repeating non-empty granules of H;

(ii) the value m;

(iii) the indexes of the first and last non-empty granules in H, if their value is not infinite.

Terminology & Notation. In the following, we term "periodic pattern" the indexes of G describing the composition of the n repeating granules of H; we call the value m "duration of the periodic pattern"; additionally, we term "frame time" the period of time spanning from the first and last non-empty granules in H. For generality, we also admit (minus and plus) infinite as legal extremes of a frame time.

Applying Property 1 to our periodic granularity PRED (i.e., substituting $H = \text{PRED}$ and $G = \text{DAY}$) we could have the following representation for PRED:

(i) $\text{PRED}(0) = \{DAY(100)\}$
$\text{PRED}(1) = \{DAY(101)\}$
$\text{PRED}(2) = \{DAY(102)\}$
$\text{PRED}(3) = \{DAY(103)\}$
$\text{PRED}(4) = \{DAY(104)\}$

(ii) $m = 28$

(iii) 100, 244

The property above is the starting point for devising our implicit representation for periodic data. We noticed that such an initial representation can be simplified along the following lines.

(a) First of all, if all granularities are expressed in terms of the bottom granularity, the bottom granularity may be left implicit. For instance, in our example, the representation of item (i) may be simplified, stating, e.g., $\text{PRED}(0) = \{100\}$ and so on.

(b) Second, contiguous granules of the bottom granularity can be more compactly represented as (convex) periods. For instance, the set of the days {100,101,102,103,104,108,109,110} can be compactly represented by {[100,104],[108,110]}. This simplification does not apply to our (simple) example.

(c) Third, it is worth remembering that any "periodic pattern" can be chosen in order to represent it. Thus, if, without any loss of generality, we adopt the convention that the chosen "periodic pattern" is the first one starting at (or immediately after) the granule "0" of the bottom granularity (DAY(0) in our example). In such a way in our representation also indexes of the granules of the quasi-periodic granularity may be left implicit.

Notice that simplifications (a) and (c) together are very important, since they allow us to keep all the indexes implicit in the representation, making it much more compact and easy.

Finally, in the case of quasi-periodic granularities, a further component must be considered into the representation, namely, the **list of the non-periodic repetitions**.

In other words, besides time periods which repeats periodically in time, we also optionally add a set of periods that do not follow such periodic pattern (the removal of some periods from the pattern is also possible, as discussed in the concluding section). Such additional periods constitute an explicit part of our representation, and model the possible positive exceptions to the periodic pattern. For such a reason, such additional periods may be during the periodic pattern (i.e., during the frame time), and also before or after it.

Remark Even in case additional non-periodic repetitions are considered, we retain the original definition for the frame time: in our approach, the frame time bounds the periodic pattern, and additional non-periodic repetitions may be inside and/or outside it.

To summarize, we propose the following implicit representation of a quasi-periodic granularity G:

Definition 10 In our approach, given a bottom granularity B, 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 a period constituting the frame time. (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 "$[i, j]$".)

Example 2 Coming back to our working shift example, PRED and PRED+ are represented in our formalism as follows:

$$\text{PRED} = \langle 28, \{100, 101, 102, 103, 104\}, \{\}, [100, 244] \rangle$$

$$\text{PRED+} = \langle 28, \{100, 101, 102, 103, 104\}, \{268, 269\}, [100, 244] \rangle$$

3.3. A relational representation of quasi-periodic data

Let us now analyze how such an abstract representation can be modelled in the relational context.

3.3.1. Data model for implicit periodic data

The abstract representation of quasi-periodic granularities we presented in subsection 3.2 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. In particular, from the theoretical point of view, such a solution 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, Per_{id} in Definition 11
- *Start* and *End* are temporal attributes (timestamps) denoting the starting and the ending points of the periods in the periodic pattern.

It is important 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 [16].*

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 [5] 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*;[2]).

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;).[3] 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 scheduled activities in an hospital. 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) patient for which the activity is performed (attribute *PatientID*), plus the temporal part. As an example, let us suppose that the prednisone treatment discussed in example 1 has to be performed on patient 'John' at the quasi-periodic time *PRED+* defined above. The corresponding representation in our extended relational model is shown in Tables 1 and 2.

[2]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

[3]Moreover, we can also deal with transaction time [5].

ActID	*Acr*	*PatientID*	VT_S	VT_E	*Per*	Per_{id}
1	*A*	*John*	100	244	28	*P*1
2	*A*	*John*	268	268	*Null*	*Null*
3	*A*	*John*	269	269	*Null*	*Null*

Table 1: `Activity` periodic relation – Implicit model

Pattern ID	*Start*	*End*
*P*1	100	100
*P*1	101	101
*P*1	102	102
*P*1	103	103
*P*1	104	104

Table 2: `Periodicity` relation – Implicit model

3.3.2. Relations between our implicit model and 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 [5]).

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

ActID	*Act*	*PatientID*	VT_S	VT_E
1	*A*	*John*	100	100
2	*A*	*John*	101	101
...	...	...	...	...
30	*A*	*John*	244	244
31	*A*	*John*	268	268
32	*A*	*John*	269	269

Table 3: `Activity_Expl` relation – Explicit model

time).

As a consequence, valid-time TSQL2 relations can be modelled (although not efficiently) as periodic relations having NULL values for the *Per* and Per_{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. 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.*

3.3.3. 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 Tables 1 and 2 above.

4. Range Queries about Implicit Periodic Data

In this and in the following sections, we focus on the issue of providing an efficient query answering methodology operating on our implicit data model. To the best of our knowledge, no *implicit* approach in the literature about user-defined periodic data has addressed such a fundamental issue. We identify three different types of *range queries*, and, for each of them, we propose efficient query answering algorithms. We also show that our approach is corrrect, in the sense that it provides the same results of standard querying on the explicit standard model for periodic data.

Range queries are particularly relevant in the context of temporal databases [21]. Specifically, the type of query we deal with is the following: given a set of quasi-periodic data (e.g., activities in the `Activity` table) and a period denoting the span of time one is interested in in the query (e.g., from DAY 110 to DAY 120), one wants to know which data holds during such a time period.[4]

In the context of periodic data, we identify three different types or range queries, depending on whether:

- one is interested in the non-temporal part of the tuples only (e.g., What activities have to be performed from DAY 110 to DAY 120 and on which patients? These queries will be called "atemporal range queries" henceforth)
- one is interested in the tuples and in their implicit time (e.g., What activities have to be performed from DAY 110 to DAY 120, on which patients, and when? Here the time in the output must be still reported following the implicit representation used

[4] *Point queries* are an easiest version of range queries, in which only a time point is taken into account in the query.

in the input data. These queries are called "implicit temporal range queries" henceforth)

- one is interested in the tuples and in their explicit time (e.g., what activities have to be performed from DAY 110 to DAY 120, and on which patients? For each activity, list all the periods they have to be performed - within the query period; Called "explicit temporal range queries" henceforth).

4.1. Atemporal Range Queries

In the following, we describe at an abstract level the algorithm for atemporal range queries. The algorithm takes in input

- a periodic relation r
- a period P_Q (the range of time requested by the query)

and gives as output a non-temporal relation containing all the (non-temporal part of the) tuples occurring (at least partially) during P_Q.

Notation. In *Atemporal_range* and in the following algorithms, we denote by $t[Atemp]$ the non-temporal part of a (periodic) tuple t, and by $t[X]$ the value of the temporal attribute X in the tuple t.

First, non-periodic tuples (i.e., "standard validity-time" tuples, if any, and the non-periodic part of periodic tuples) are considered. If the period starting from VT_S and ending in VT_E intersects the query period P_Q, they (or, better, their non-temporal part) are reported in output. Then, periodic tuples are taken into account. While *Intersects* implements standard check of intersection between two periods, in the case of periodic tuples (i.e., tuples such that $P_{ID} \neq NULL$) the valid time is expressed only in an implicit way, so that *Check_Periodic_Intersection* has to be called.

Input: r: periodic relation, P_Q: period
Output: r': atemporal relation
$r' \leftarrow \oslash$;
$r_{aper} \leftarrow$ Select * From r Where Per_{ID}=NULL;
$r_{per} \leftarrow$ Select * from r Where $Per_{ID} \neq$ NULL;
foreach *tuple* $t \in r_{aper}$ **do**
  if *NOT* $(t[Atemp] \in r')$ **then**
    Let P_t be the period $[t[VT_S], t[VT_E]]$;
    if $Intersects(P_t, P_Q)$ **then**
      $r' \leftarrow r' \cup \{t[Atemp]\}$
    end
  end
end
foreach *tuple* $t \in r_{per}$ **do**
  if *NOT* $(t[Atemp] \in r')$ **then**
    if $Check\_Periodic\_Intersection(t, P_Q)$ **then**
      $r' \leftarrow r' \cup \{t[Atemp]\}$
    end
  end
end
return (r')

Algorithm 1: Atemporal range queries

Notice that the algorithm is optimised in such a way that:

- Easier cases (i.e., cases in which the valid time is not implicit) are considered first
- Whenever a tuple is inserted in the output relation, its *value-equivalent* tuples (i.e., tuples with the same values for the non-temporal attributes) are no more analyzed (since they couldn't add anything to the output relation)

Check_Periodic_Intersection has as input a periodic tuple (such that $Per_{ID} \neq NULL$), and the query period P_Q, and checks whether there is an intersection.

Input: (t: periodic tuple; P_Q:period)
Output: boolean
let P_t be the period $[t[VT_S], t[VT_E]]$;
if $Intersects(P_t,\ P_Q)$ **then**
    Let $P' \leftarrow P_t \bigcap P_Q$;
    if $duration(P') \geq t[Per]$ **then**
        return (TRUE);
        else
        $Pset \leftarrow$
        $get\_periods(t[Per_{ID}],$ `Periodicity`);
        if
        $G\_Intersects(circular\_module(P', t[Per]),$
        $circular\_module(Pset, t[Per]))$ **then**
            return (TRUE);
            else
            return (FALSE);
        end
    end
end

Algorithm 2: Check Periodic Intersection

First, the Frame Time $P_t = [t[VT_S], t[VT_E]]$ is checked: the data can be in the answer just in case the Frame Time intersects the query interval P_Q. In such a case, however, we have to further check if there is an intersection between some of the repetitions and the query interval. Henceforth, only the part of the query period that intersects the Frame Time must be considered (i.e., P', which is set to $P_t \bigcap P_Q$), since the other part cannot obviously overlap any of the repetitions. A first shortcut check might be done by comparing the temporal duration of P' to the duration of the repetition pattern $t[Per]$. If P' is longer, then it will obviously contain at least one of the repetitions, so that the check evaluates to TRUE (in the following, we will call "OPT" such an optimization of the algorithm). Otherwise, since periods repeat regularly, with period $t[Per]$, we have to check for intersections "modulo $t[Per]$". In fact, both the periods in the periodic pattern (identified by $t[Per_{ID}]$, and retrieved from the `Periodicity` system table by the get_periods function) and P' must be "shifted" onto a common period. For the sake of simplicity and efficiency, we choose to shift them to the period identified by the first possible occurrence of the periodic pattern after the reference time. This operation is performed by the "circular_module" operation, and then intersections can be checked.[5] The "*circular_module*" operation is basically a standard module operation which operates on the endpoints of all the input periods, with the caution that, since the pattern is periodic, it must be interpreted in a "circular" fashion with respect to the common period. Finally, notice that *G_Intersects* is a trivial generalization of the *Intersects* predicate, in case the second argument is a set of periods.

4.2. Implicit temporal range queries

The algorithm for implicit temporal range queries is quite an easy adaptation of the algorithms above. In particular, the only difference is the fact that also the temporal part of the tuples must be reported in the output. Therefore, as regards the non-periodic tuples, value-equivalent tuples (with different values for VT_S and VT_E) must be reported in the output. Periodic tuples

[5] Alternatively, all the periods identified by $t[Per_{ID}]$ could be shifted towards P'. However, such a procedure is less efficient.

are analysed independently of whether value-equivalent tuples are already part of the output: if their valid time intersects the query period, they are reported in the output, with their implicit temporal part. Notice that the values of the VT_S and VT_E temporal attributes are modified, in such a way that only the part of the original frame time that intersects the query period is reported in the output. On the other hand, the input pattern that implicitly represents the periodicity of the tuples is reported unchanged in the output.

4.3. Explicit temporal range queries

The algorithm for explicit temporal range queries is an adaptation of the algorithms above. The only conceptual shift is the following: whenever a periodic tuple has to be reported in the output, it has to be converted into a set of value-equivalent tuples, having as non-temporal part the non-temporal part of the periodic tuple, and as temporal part (i.e., as values of the VT_S and VT_E components) one of the periods obtained by generating explicitly all the periods that intersect the query period that are denoted by the implicit representation. Basically, such an algorithm interprets the periodic pattern as a circular list, and generates only the periods which are included in the intersection between the frame time and the query period. However, several cases have to be taken into account, and several optimizations can be used.

4.4. Correctness of the method

The algorithms above operate on the implicit model, to answer range queries. Our method is correct, in the sense that it provides the same results provided by the explicit approach (on explicit data). Only the algorithms for atemporal range queries and explicit temporal range queries have been considered here, since implicit temporal range queries have no correspondent in the explicit approach (since the explicit approach cannot provide implicit output).

Property 3. *[Correctness] Range query answering algorithms, operating on the extended temporal model, are correct with respect to the explicit approach.*

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.

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.

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 500MB. 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 [32].

We run two different sets of experiments, on two different data sets. In the first, we compare our approach to the explicit one, showing that we outperform current explicit approaches, even of a factor of 20 as regards the Disk I/O. The first set of experiments also shows that there is a trade-off: our implicit approach save space and physical disk accesses, at the prize of expending relatively more CPU time. In the second set of experiments, we explore such a trade-off, showing that the results of our approach improve when the ratio between the size of the implicit representation and the size of the explicit one becomes smaller.

5.1. Data sets

For the testing, we have chosen to run two different types of experiments:

- One type of experiments takes advantage of previous experiences in the medical informatics domain.
- The second type of experiments takes into account completely ad-hoc data, artificially generated in order to study the impact of different parameters (and, in particular, of the explicit/implicit ratio - see subsection 5.5).

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 [11]. 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).

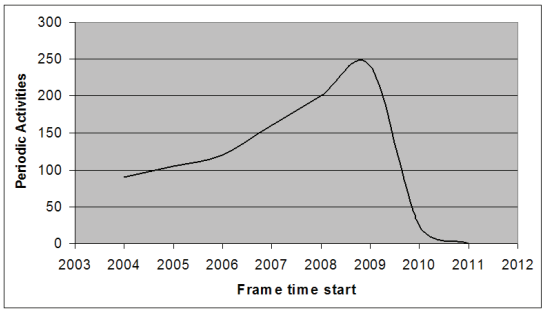

Figure 2: Average number of new periodic activities per day over the Frame time start

Privacy motivations impose that we do not have currently available real medical data. However, based on our experience in the area, 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**
- Distribution of the frame time – see Figure 2 where we assume NOW=January 1, 2009; It can be seen that during the past the number of new activities

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.

is slightly increasing, while future data are reducing (in fact, it is likely to know new 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.

Despite the fact that 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. Data Indexing

Different indexing methods can be applied in order to retrieve data avoiding to scan the full data space. When more than one dimension exist, as in the case of temporal interval data (e.g., when both a starting and an ending time are present for each tuple), different access methods have been presented in the literature and some of them have been recommended for handling temporal data [27], [28], [29]. The effectiveness of the majority of these index structures has been theoretically evaluated [30].

Since it has been shown in the literature that the RI-Tree [31] 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, in the experiments we used RI-tree also to index the valid times in the explicit approach.

5.3. Experiment: description

In order to carry on the experiments, the same periodic activities concerning hospitalized 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 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 [33]. 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 [31].

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 system views. Space usage for Tables and Index structures are collected from the data dictionary view User_Segments.

5.4. Range Queries: Results and Analysis

First, let us describe our results as regards the treatment of range queries.

In Tables 7, 8, 9 we show physical disk I/O's, CPU time and response time for atemporal range queries considering two parameters: query duration and answer size. Different range queries duration are considered in order to investigate the effect of execution of the circular-module function. Different answer sizes are considered to see the effect of clustering the data, aging of database buffers and effect of trade between Physical disk I/O and CPU usage. Each value in the tables (here and in the following) is the average value obtained after ten runs and every run was on empty database buffer cache in order to avoid effect of logical read of data. Moreover, the temporal location of the query period along the timeline has been chosen randomly in order to eliminate the effects of distribution of data.

The experimental results clearly show the advantages of our implicit approach. For instance, considering 168 hours (query duration) and atemporal range queries with answer size 29,455, the implicit approach requires about 1/20 of Disk I/O and CPU time and about 1/35 response time with respect to the explicit method.

Looking at the experimental results in more details, one can notice that, in case the duration of the query is short (e.g., 1 hour) and the answer size is small (e.g., 580; See Table 7), the explicit approach behaves better than our implicit one, especially as regards CPU time. This fact is basically caused by the overhead of the implicit approach due to the evaluation of circular-module onto the periods retrieved from the `Periodicity` table, which (given the small answer size), mostly do not contribute to the result.

Obviously, disk I/O, CPU time and response time of both the implicit and the explicit approach increase when the answer size increases. However, the increase of the implicit approach is much slower, especially as regards disk I/O and response time. For instance, with a short query duration (1 hour) and answer size 17,738 (see Table 7), the implicit approach requires about 1/3 disk I/O (due to the fact that less records need to be fetched) and 3 times CPU time (due to the computation of the circular-module) of the explicit approach, and the overall balance (response time) is about three times better than the explicit approach.

The advantages of our implicit approach become larger and larger in case the query period increases. In such a case, in fact, thanks to the optimization OPT (which applies whenever the duration of the repeti-

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

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						19,427.83

Table 6: Space requirements for the Explicit model

Answer size	*Implicit model*			*Explicit model*		
	Disk I/O	CPU	Response time	Disk I/O	CPU	Response time
580	2,273	151	2	1,002	28	1
2,068	6,049	477	5	3,031	73	3
3,721	8,304	753	8	6,336	168	8
7,471	10,831	1,431	12	12,872	315	30
17,738	12,364	3,186	32	31,904	930	154

Table 7: Average CPU usage, Physical disk I/O and Query response time for 1 Hour atemporal range queries depending on the answer size

tion pattern is smaller than the query period) the implicit approach needs to fetch less periods from Table `Periodicity`, and, consequently, also to perform less computations of the circular-module function. For instance, with query duration 168 hours (one week), and with answer size 2,887, the implicit approach uses 1/4 disk I/O, 1/3 CPU time, and 1/17 response time of the explicit approach, and with answer size 29,455 the advantage arise to 1/20 (I/O), 1/20 (CPU), and 1/35 (response time).

The case of temporal range queries is quite similar. The only qualitative difference is due to the fact that, in the implicit approach, all the periods intersecting the query period must be generated. This means that the optimization OPT above does not apply to temporal range queries. As a consequence, results are not sensitive of the query period, except for the fact that it may influence the answer size. Results, for different answer sizes, are reported in Table 10. Table 10 shows that there is a trade-off between CPU computation and disk I/O: our implicit approach is more CPU demanding, but saves on disk I/O. As in the case of atemporal range queries, the benefits of our implicit approach become more evident when answer size increases. For instance, with answer size 17,738, the implicit approach requires 1/3 disk I/O and 3 times CPU of the explicit one, and the total response time is 1/5.

In other words, there is clearly a trade-off between storage requirements and computation: our implicit approach save space and physical disk accesses, at the prize of spending relatively more CPU time. This is in itself a positive result, since, in the area of databases, physical disk accesses are considered to be the bottleneck (while, e.g., CPU time might be reduced through the introduction of more powerful CPU's, and/or with an extension of the number of CPU's), and therefore physical disk accesses is the most important aspect to be taken into account [34].

The core result emerging from the experiments described so far is that the overall balance of our approach with respect to the explicit one is widely positive: we gain in response time, and our gain increases with the increase of the size of the answer.

In the following subsection, we propose additional experiments, in order to further explore the above trade-off.

5.5. Range Queries: Exploring the trade-off

To summarize, since our approach proposes an implicit representation of periodic data, it saves space and disk accesses, at the prize of being more computation-demanding. In the previous subsection, we have drawn a real-world experiment in the hospitalization context, which has shown the advantage of adopting our approach. In other domains, such as manufacturing, few activities (which can be specified through very simple periodic pattern, whose duration may be very small – sometimes in the order of seconds) can be repeated for very long frame times (sometime in the order of years). Clearly, our approach is even more advantageous in such domains. In other words, the greater is the number of periodic repetitions of data, the greater is the gain of our approach. In this subsection, we draw additional experiments to substantiate such a claim.

Specifically, we have generated artificial data, in such a way that we get different values for the ratio between the size of the implicit representation and the size of the corresponding explicit one (henceforth called "explicit/implicit" ratio; in this experiment, we do not take into account the size of the indexes). For in-

Answer size	*Implicit model*			*Explicit model*		
	Disk I/O	CPU	Response time	Disk I/O	CPU	Response time
580	966	88	1	2,726	85	2
2,068	2,466	192	3	8,379	255	6
3,721	3,368	277	4	9,851	308	9
7,471	5,361	683	8	13,028	703	32
17,738	8,896	1,134	22	41,854	1,236	162

Table 8: Average CPU usage, Physical disk I/O and Query response time for 24 Hour atemporal range queries depending on the answer size

Answer size	*Implicit model*			*Explicit model*		
	Disk I/O	CPU	Response time	Disk I/O	CPU	Response time
2,887	1,838	200	3	7,970	601	52
4,620	2,114	260	5	12,913	1,232	116
29,455	9,490	1,375	26	183,073	26,460	944

Table 9: Average CPU usage, Physical disk I/O and Query response time for 168 Hour atemporal range queries depending on the answer size

Query duration	*Implicit model*			*Explicit model*		
	Disk I/O	CPU	Duration	Disk I/O	CPU	Duration
1	156	309	4	1,688	66	11
24	646	3,143	12	17,292	705	92
168	1,246	22,752	45	156,231	21,460	912

Table 10: Average CPU usage, Physical disk I/O and Query response time for explicit temporal range queries depending on range query duration

Approach	Ratio	Disk I/O	CPU	Resp. Time
Impl.	1	57	100	4
Expl.	1	52	14	1
Impl.	5	96	76	5
Expl.	5	814	109	8
Impl.	50	331	161	5
Expl.	50	2,696	280	16
Impl.	100	347	221	5
Expl.	100	5,778	542	26

Table 11: Tradeoff between the implicit (Impl.) and the explicit (Expl.) approaches, considering different "explicit/implicit" ratios

stance, the value 50 for the ratio copes with the case in which the explicit representation is 50 times the size of the implicit one). Implicit tables `Activity` and `Periodicity` for all ratios in our experiments contain 10,000 and about 50,000 records respectively (For instance, the `Activity_Expl` table contain about 3 millions rows for explicit/implicit ratio of 50 (50*(10,000 + 50,000))). We have generated the data for the desired ratio's simply by changing the duration of the Frame time and the duration of the repetition pattern. As a consequence, in our experiments the CPU and response time of the implicit approach are quite non-sensitive with respect to the changing of the ratio (since the dimension of the implicit tables remain quite constant). On the other hand, the explicit method is significantly influenced by the increasing ratio. This is due to fact that the number of rows in the tables of the explicit approach linearly increase with the ratio (while in the implicit approach we basically maintain the same rows, only increasing the duration of the frame time). All the experiments in table 11 have been run with a query duration of 168 hours.

The experiments have shown that already at Explicit/Implicit ratio of 5 our implicit method starts to gain significant advantages as regards both disk I/O and response time. Of course, advantages become greater and greater when the ratio increases. With ratio 100, the implict approach needs 1/15 disk I/O, less than 1/2 CPU, and less then 1/5 response time of the explicit approach.

5.6. 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:

- Implicit temporal range queries cannot be coped with in the extensional approach, since it does not provide any implicit description of periodic data; only the whole list of repetitions can be provided as output;
- 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).

6. Related works

Several medical activities, as well as many other human activities (consider, e.g., office activities, scheduling of train, airplanes, lessons, ...), are scheduled at periodic time. 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 [35] 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 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), and a large number of approaches has been defined to such a purpose (e.g., the survey by Tuzhilin and Clifford [36], dating back to 1995, focuses on the Database area, and takes into account 34 different approaches). Periodic data play an important role in Databases, so that, for instance, a specific entry (see [37]) has been devoted to such a topic in the out-coming Encyclopedia of Database Systems by Springer [38]). In the Encyclopedia [37], three main classes of Database (implicit) approaches to user-defined periodicities have been identified: *Deductive rule-based* approaches, using deductive rules (e.g., [39] and approaches in classical temporal logics), *constraint-based* approaches, using mathematical formulae and constraints (e.g., [13]), and *algebraic* (also termed *symbolic*) approaches, providing a set of "high-level" and "user-friendly" operators (e.g., [14], [15], [16], [40], [41], [42], [43] [44], [45], [46]). A comparison among such classes approaches is out of the scope of this paper (the interested reader is referred, e.g., to [37]). 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.

In the following, we focus on the approaches that are more closely related to our one. The idea of proposing an implicit representation formalism based on the "consensus" definition of periodic granularities has been pursued also by Ning et al [15]. 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. In [47] it is shown how the user-defined periodic granularities in Ning et al.'s formalism can be translated into the notation used for the "consensus" definition (in [16]). In particular, also the algebraic operations of union, intersection and difference have been considered. On the other hand, neither [15] nor [47] have focused on the core issues of representing periodic data in the relational context, and of coping efficiently with range queries (which, on the other hand, constitute the central focus we have faced in our implicit approach). Terenziani 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. 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 [14]. Kabanza et al. [13] 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 [13], 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 [41] 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.

On the other hand, periodic data have also been coped with in the Object Oriented context. For example, [48] defined an extension of SQL3 based on the temporal type of *periodic elements*. Periodic elements consist of both an aperiodic part (represented by an extensional set of time periods) and a periodic part, represented by the repetition pattern and the period of repetition (roughly corresponding to our frame time). They also defined the set-operations of union, intersection, complement and difference, which are closed with respect to periodic elements, and used them in the definition of a periodic temporal extension of object oriented SQL (SQL3).

In summary, it is worth noticing that none of the above relational approaches is based on the "consensus" definition of periodic granularity, and none of them is a consistent extension of the "consensus" TSQL2 model.

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 and 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.

7. Conclusion and Future Work

Periodically-repeated data play an important role in medicine (for instance, in the scheduling of periodic activities on patients). In this paper, we apply Artificial Intelligence symbolic manipulation techniques to the application context of relational databases to propose a new approach to cope with periodic data in 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 [17] and its extension to cover periodic granularities in [16],

which is a “consistent extension” of TSQL2’s one [5].

- we have taken into account range queries, providing query answering algorithms for them; the algorithms operate as much as possible in a symbolic way on the implicit representation, and are correct, in the sense that they provide (in a more efficient way) the same results that would be obtained by a direct treatment of the explicit data.

- we have developed an extensive experimentation of our model and methodology, showing that our “implicit” and symbolic approach overcomes the performance of traditional “explicit” approaches both in terms of space and disk I/O’s, and in terms of answer response time. Moreover, we have also analysed to what extent our implicit approach is advantageous, depending on the “explicit/implicit” ratio.

In this paper, we have focused on periodic patterns of events, but we have also considered the possibility of having ”positive” exceptions (i.e., additional events that are not part of the periodic pattern). Indeed, this is in the style of database approaches, in which only facts that actually occur are considered (while there is no mention of facts that never occurred - consider, e.g., the definition of quasi-periodic granularity in the literature [16], and [48]). However, in the context of periodic repetitions, ”negative” exceptions may also be relevant. For instance, a treatment may be scheduled each day for a month, with the exception of a specific day. From the representation point of view, the extensions required to cope with ”negative” exceptions are minimal. We can still regard the periodic pattern (represented in an implicit way) as the ”core” of the representation, and cope in an explicit way with two types of exceptions: the list of positive exceptions and the list of negative ones. (Of course, by definition, negative exceptions must be during the frame time of the periodic pattern). In the relational representation, negative exceptions can be represented by explicit tuples. However, an additional attribute must be considered, in order to distinguish ”positive” versus ”negative” exceptions. Modifications must be also operated on the query answering algorithms. For instance, as regards atemporal range queries, the algorithm *Check_Periodic_Intersection* must be modified. First of all, the optimization OPT must be removed, since the fact that the query period is longer than the duration of the repetition pattern does no longer imply that there is at least one repetition in the query period (indeed, all such repetitions may be explicitly excluded as negative exceptions). Additionally, *G_Intersects* must be modified in order to take into account also negative exceptions.

Finally, 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 [14] , [15], [16], [43], [40], [41] 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

[1] SNODGRASS, R. T. 1999 Developing Time-Oriented Database, Applications in SQL Morgan Kaufmann Publishers, Inc., San-Francisco.

[2] A.K. DAS AND M.A. MUSEN A foundational model of time for heterogeneous clinical databases, In *Proc. AMIA'97*, 106-110.

[3] R.JAMES AND C. GOBLE Survey and critique of time and medical records, In *Proc. Medinfo'95* , 271-275.

[4] GULTEKIN , OZSOYOGLU AND R.T. SNODGRASS Temporal and Real-Time Databases: A Survey, *IEEE Transactions on Knowledge and Data Engineering* ,7(4), 1995, 513-532.

[5] SNODGRASS, R. T. 1995. *The TSQL2 Temporal Query Language*. Kluwer Academic.

[6] A.K. DAS AND M.A. MUSEN A temporal query system for protocol-directed decision support *Methods Inf. Med* ,33(4), 1994, 358-70.

[7] M.J. O' CONNOR, S. TU, AND M.A. MUSEN The Chronus II Temporal Database Mediator In *Proc. AMIA'02*, 567-571.

[8] P. TERENZIANI, R SNODGRASS Reconciling Point-based and Interval-based Semantics in Temporal Relational Databases: A Proper Treatment of the Telic/Atelic Distinction. IEEE Transactions on Knowledge and Data Engineering, 16(4),540-551, Aprile 2004.

[9] P. TERENZIANI, R.T. SNODGRASS, A. BOTTRIGHI, G. MOLINO, M. TORCHIO, Extending Temporal Databases to Deal with Telic/Atelic Medical Data, Artificial Intelligence in Medicine 39(2), Elsevier, 113-126, 2007.

[10] G.DUFTSCHMID, S. MIKSCH AND W. GALL Verification of temporal scheduling constraints in clinical practice guidelines. *Artificial Intelligence in Medicined* ,25(2) (2002), 93-121.

[11] ANSELMA, L., TERENZIANI, P., MONTANI, S., AND BOTTRIGHI, A. 2006. Towards a comprehensive treatment of repetitions, periodicity and temporal constraints in clinical guidelines. *Arificial Intelligence in Medicine 38,* 2, 171–195.

[12] A. BOTTRIGHI, L.GIODANO, G. MOLINO, S. MONTANI, P. TERENZIANI, M. TORCHIO Adopting model checking techniques for clinical guidelines verification, Artificial Intelligence in Medicine 48(1), 1-19, 2010.

[13] KABANZA, F., STEVENNE, J.-M., AND WOLPER, P. 1995. Handling infinite temporal data. *Journal of Computer and System Sciences 51*, 3–17.

[14] TERENZIANI, P. 2003. Symbolic user-defined periodicity in temporal relational databases. *IEEE TKDE 15,* 2, 489–509.

[15] NING, P., WANG, X., AND JAJODIA, S. 2002. An algebraic representation of calendars. *Annals of Mathematics and Artificial Intelligence 36,* 1-2, 5–38.

[16] BETTINI, C. AND DE SIBI, R. 2000. Symbolic representation of user-defined time granularities. *Annals of Mathematics and Artificial Intelligence 30,* 1-4, 53–92.

[17] BETTINI, C., DYRESON, C., EVANS, W., SNODGRASS, R..T., AND 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*, 406–413.

[18] SNODGRASS, R. T. AND AHN, I. 1985. A Taxonomy of Time in Databases. In *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data*, S. B. Navathe, Ed. ACM Press, 236–246.

[19] DYRESON, C. E., SNODGRASS, R. T., AND FREIMAN, M. 1995. Efficiently Supporting Temporal Granularities in a DBMS. Tech. Rep. TR 95/07.

[20] BETTINI, C., JAJODIA, S., AND WANG, S. 2000. *Time Granularities in Databases, Data Mining, and Temporal Reasoning*. Springer Verlag.

[21] TSOTRAS, V. J., JENSEN, C. S., AND SNODGRASS, R. T. 1998. An Extensible Notation for Spatiotemporal Index Queries. *ACM SIGMOD Record 27,* 1, 47–53.

[22] STANTIC, B., TERENZIANI, P., SATTAR, A., BOTTRIGHI, A. and GOVERNATORI, G. Towards an implicit treatment of periodically-repeated medical data In Proceedings of the 13th World Congress on Medical Informatics 2010, 1131–1135.

[23] CODD, E. 1971. Relational completeness of data base sublanguages. In *Courant Computer Science Symposia 6, Data Base Systems*. 24–25.

[24] L. EDWIN MCKENZIE, J. AND SNODGRASS, R. T. 1991. Evaluation of Relational Algebras Incorporating the Time Dimension in Databases. *ACM Computing Surveys (CSUR) 23,* 4, 501–543.

[25] JENSEN, C. S. AND SNODGRASS, R. T. 1996. Semantics of Time-Varying Information. *Information Systems 21,* 4, 311–352.

[26] GAO D. AND JENSEN C. AND SNODGRASS R.T. AND SOO M.D.. 2005. Join Operations in Temporal Databases. *VLDBJ 14*, 2–29.

[27] KUMAR, A., TSOTRAS, V. J., AND FALOUTSOS, C. 1997. Designing access methods for bitemporal databases. *University of Maryland at College Park; Report No. UMIACS-TR-97-24*.

[28] BOZKAYA, T. AND OZSOYOGLU, Z. M. 1998. Indexing valid time intervals. In *Database and Expert Systems Applications*. 541–550.

[29] KOLLIOS, G. AND TSOTRAS, V. J. 2002. Hashing methods for temporal data. *IEEE Transactions on Knowledge and Data Engineering 14,* 4, 902–919.

[30] SALZBERG, B. AND TSOTRAS, V. J. 1999. Comparison of Access Methods for Time Evolving Data. *ACM Computiong Surveys 31,* 1.

[31] KRIEGEL, H.-P., PTKE, M., AND SEIDL, T. 2000. Managing intervals efficiently in object-relational databases. *Proceedings of the 26th International Conference on Very Large Databases*, 407–418.

[32] HELLERSTEIN, J., KOUTSUPIAS, E., AND PAPADIMITRIOU, C. 1997. On the Analysis of Indexing Schemes. *16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*.

[33] KRIEGEL, H.-P., POTKE, M., AND SEIDL, T. 2001. Object-relational indexing for general interval relationships. In *Proc. 7th Intl Symposium on Spatial and Temporal Databases (SSTD01)*.

[34] HELLERSTEIN, J. M., NAUGHTON, J. F., AND PFEFFER, A. 1995. Generalized Search Trees for Database Systems. In *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, Zurich, Switzerland*. 562–573.

[35] SOO, M. AND SNODGRASS, R. 1993. Multiple calendar support for conventional database management systems. In *Proc. Int'l Workshop on an Infrastructure for Temporal Databases*.

[36] TUZHILIN, A. AND CLIFFORD, J. 1995. On periodicity in temporal databases. *Information Systems 20,* 8, 619–639.

[37] TERENZIANI, P.2009. Temporal periodicity. In *Encyclopedia of Database Systems*. Springer Verlag, Ling Liu and M. Tamer zsu Editors.

[38] LIU, L. AND ZSU, M. T. 2009. *Encyclopedia of Database Systems*. Springer Verlag.

[39] CHOMICKI, J. AND IMIELINSKI, T. 1993. Finite representation of infinite query answers. *ACM ToDS 18,* 2, 181–223.

[40] LEBAN, B., MCDONALD, D., AND FORSTER, D. 1986. A representation for collections of temporal intervals. In *Procs. of AAAI'86*. 367–371.

[41] NIEZETTE, M. AND STEVENNE, J.-M. 1992. An efficient symbolic representation of periodic time. In *Procs. of CIKM*.

[42] P. TERENZIANI Integrated Temporal Reasoning with Periodic Events. Computational Intelligence 16(2), 210-256, May 2000.

[43] EGIDI, L. AND TERENZIANI, P. 2005. A flexible approach to user-defined symbolic granularities in temporal databases. In *Procs. of ACM SAC'05*. 592–597.

[44] L. EGIDI, P. TERENZIANI, A lattice of classes of user-defined symbolic periodicities Proc. 11th International Symposium on Temporal Representation and Reasoning, IEEE press, 21-27, Tatihou Island, Normandie, France, 2004.

[45] L. EGIDI, P. TERENZIANI A Mathematical Framework for the semantics of symbolic languages representing periodic time, Annals of Mathematics and Artificial Intelligence 46(3), Springer Verlag, 317-347, 2006.

[46] L. Egidi, P. Terenziani. A modular approach to user-defined simbolic periodicities, Data & Knowledge Engineering 66(1), Elsevier, 163-198, 2008 (DOI information: 10.1016/j.datak.2008.02.003).

[47] BETTINI, C., MASCETTI, S., AND WANG, X. 2004. Mapping Calendar Expressions into Periodical Granularities. In *Procs. TIME 2004*. 96–102.

[48] KURT, A. AND OZSOYOGLU, M. 1995. Modelling and Querying Periodic Temporal Databases. In *Workshop of the 6^{th} International Conference on Database and Expert Systems Applications (DEXA)*. 1 –133.