

Modeling and Querying Periodic Temporal Databases

Atakan Kurt

Meral Ozsoyoglu

Computer Eng. and Sci. Department
Case Western Reserve University
Crawford Hall, 5th floor
Cleveland OH 44116

Abstract

Temporal events of periodic nature occur very frequently in a variety of application domains such as scheduling, planning, forecasting, scientific, multimedia, real-time, and active databases. Existing temporal data models support only aperiodic events with a few exceptions. We present an approach based on a new periodic temporal type called *periodic element* to model periodic events in temporal databases, and illustrate its suitability by extending object-oriented SQL (SQL3). Our paradigm is compatible with existing aperiodic data models, i.e., it can be used to extend them seamlessly. Periodic elements are capable of representing not only *aperiodic* and *strictly-periodic* temporal events but also *partially-periodic* events as well. They can represent *absolute* and *relative* events uniformly and are closed under the set theoretic operators with a set of optimization rules. We provide a canonical form for efficient storage. Periodic elements can be easily incorporated in a object oriented language as a type hierarchy. Our approach can be used to model domains involving periodic events such as calendars, periodic multimedia events as well.

Keywords: Temporal databases, periodic temporal types, object-oriented databases, SQL, SQL3, temporal query languages, periodic events, aperiodic events, partially-periodic events.

1. Introduction

Many real world applications such as scheduling, planning, forecasting, time-management, time-series, scientific, multimedia, real-time, and active databases, banking, law, medical records, accounting, process control, inventory control, geographical information systems deal with periodic events. The flight, bus, and train schedules are examples of periodic events. In real-time control systems and real-time databases, the

outputs of a system must be analyzed/stored periodically and the inputs to such a system must be provided in a periodical fashion. The set of tasks a robot performs is an example of a periodic event in a real-time system. A multimedia database stores video and audio sequences which must be synchronously played out. Active databases are characterized by the *on-event if-condition do-action* rules. These rules must be evaluated periodically to check if any of the conditions are satisfied. The *if-condition* clause may include a periodic expression such as *every 10 minutes*, *every Monday* or *the first day of every month*. Scheduling workers in an organization involves periodic events. The following example illustrates the need for modeling periodic events.

Example 1: Consider a company employing full-time and part-time employees. The work hours of full-time employees can be maintained by a conventional temporal database system, since they can be represented by intervals [2] or temporal elements [5]. The work hours of part-time employees can't be stored in a conventional temporal database, because employees have periodical work hours like daily, weekly, or monthly. For training purposes, some full-time employees must switch to a different department in every 3 months. The work hours of employees rotating periodically among departments can't be stored and modeled in aperiodic temporal databases either. Among other conventional temporal queries (queries about the past), the management wants to use the periodic nature of events to plan the future activities, the number of part-time workers at a past or future date, the employees working in a department at a given future date, etc.

The work schedule of full time employees in this application can be modeled by aperiodic temporal databases such as [GaVa 85, Ta 86, SeSh 87, SuCh 91, Ta 91, RoSe 91, WuDa 92, Ah 93, GaBh 93, Ta 93, Ga 93, Sn 94]. However the work schedule of part-time

employees or rotating full-time employees can not be modeled in these models because these models do not have any periodic type or construct to handle such events, and can't reason about periodic events. In this case, periodic events must be directly maintained and interpreted by application programs in an ad hoc manner. There is a need for defining a periodic type to allow users to maintain and query periodic data similarly to aperiodic data for the application domains mentioned above.

The rest of the paper is organized as follows. In the next section, we present *periodic elements* capable of representing periodic events and define different types temporal events using periodic elements. We discuss the SQL3 proposal briefly in Section 3 and present the periodic temporal extension of SQL3 for periodic temporal data. SQL3 is extended by introducing a type hierarchy into the language while keeping the SQL3 syntax unchanged. We illustrate the extension with a set of example queries in Section 3.4. The temporal expressions involving periodic events can be optimized and stored in an efficient manner by transforming temporal values into canonical form as discussed in Section 4. We compare our work with related research in Section 5. Finally we conclude in Section 6.

2. Periodic Element as a Temporal Type

Periodic elements are formally based temporal elements. We will briefly review aperiodic types here. A detailed presentation of aperiodic types and periodic elements can be found in [11]. There are set of constants defined as follows: \emptyset and ∞ stand for empty temporal values and positive infinity on the time line. *now* represents the current time. We assume that time starts at instant 0 for simplicity.

An *instant* t is a specific point on the physical time line. Instants represent temporal events happening instantaneously. In a *discrete* model, instants are represented by natural numbers. In *dense* time models, instants are isomorphic to the rationals or the reals. *Continuous* models of time are isomorphic to the reals, not allowing gaps between instants as in models isomorphic to the rationals. An *interval* $[b, e]$ is a specific duration of time beginning at instant b and ending at instant e inclusively. Intervals $(b, e]$, $[b, e)$, and (b, e) with open endpoints are defined similarly. Every instant t is an interval $[t, t]$ by default. The temporal comparison operators *before*, *after*, *meets*, *overlaps*, etc. between intervals are used to determine relationships between temporal events [2]. Intervals represent events happening over a single duration of time and are used in many temporal models. Events

happening over more than one duration of time can't be represented by a single interval. They are represented by a set intervals called *temporal elements* [5]. A temporal element N is given by listing the intervals in it:

$$N = ([b_1, e_1], [b_2, e_2], \dots, [b_k, e_k]) \text{ for } k \geq 1.$$

Parenthesis can be omitted, when there is no confusion. By definition, every interval $[b, e]$ is a temporal element. The set of temporal elements are closed under the set theoretic operations of *union* \cup , *intersection* \cap , and *complementation* \sim with T (the set of all time instants) as its maximum element and \emptyset as its minimum element [5] meaning that *and*, *or*, and *not* of natural language expressions can be expressed in a language model based on temporal elements. These operators are shown graphically in

Figure 1.

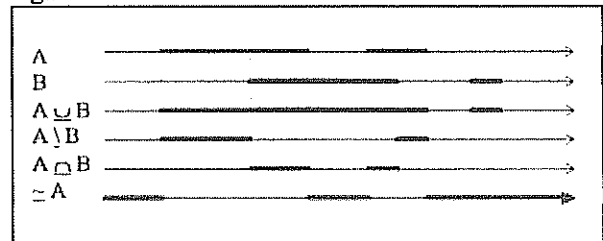


Figure 1: The Set theoretic operators on temporal elements.

Among other operations like *distance*, *first_interval*, *last_interval*, etc. on temporal elements, temporal transformations are defined as well. For a temporal element $N = ([t_1, t_2], \dots, [t_{n-1}, t_n])$, transformations are given as follows:

- *translate* $(([t_1, t_2], \dots, [t_{n-1}, t_n]), d) = ([t_1+d, t_2+d], \dots, [t_{n-1}+d, t_n+d])$ for $d \in I$.
- *scale* $(([t_1, t_2], \dots, [t_{n-1}, t_n]), s) = ([t_1, t_1+(t_2-t_1)*s], \dots, [t_1+(t_{n-1}-t_1)*s, t_1+(t_n-t_1)*s])$ for $s \in R$.

Function *translate* translates a temporal event on the time line to the right or the left by d time units. Function *scale* scales up or down a temporal event by a factor of s . The beginning of the event is not changed by the scale factor, however each interval in the element is scaled. For $\alpha = ([3, 6], [8, 9])$, $scale(\alpha, 2) = ([3, 9], [13, 15])$. Though able represent more complex events than intervals, temporal elements can represent only events happening aperiodically and over a finite number of intervals. To model periodic events, we introduce periodic elements. A *periodic element* consists of three components: *an aperiodic part*, *a periodic part and a period*:

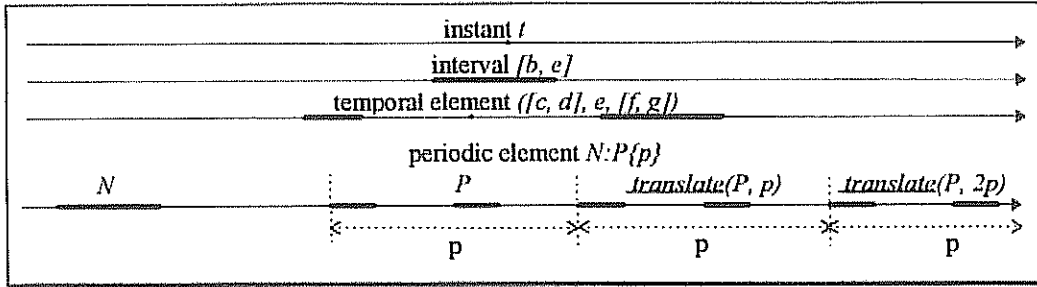


Figure 2: The graphical representation of temporal types.

1. *Aperiodic part* is a temporal element denoting the initial irregular temporal pattern of the temporal event,
2. *Periodic part* is also a temporal element denoting the regular indefinitely-many-repeating temporal pattern of the temporal event, and
3. *Period* is a duration of time denoting the cyclic duration of the *periodic part*.

Definition 1 (Periodic Element): A triplet of two temporal elements $N = (I_1, I_2, \dots, I_n)$ and $P = (J_1, J_2, \dots, J_m)$, and a duration $p \in I$ is called a periodic element $N:P\{p\}$ where

1. $p \geq 0$,
2. I_n before $J_1 = \text{true}$ ¹,
3. $p \geq \text{last_instant}(J_m) - \text{first_instant}(J_1)$ ².

The first condition above insures that the periodic event being represented extends into positive infinity, i.e., the future. Negative values may be considered for events happening backwards in time and are not considered here. The second condition states that the aperiodic part of the event must happen *before* the periodic part to distinguish between the two. This is an essential property so that events first happening aperiodically and then happening periodically can be modeled. The third condition asserts that the *period* must be at least as long as the length of the periodic part of the event. Formally a periodic element is the union of its aperiodic N and repeating periodic part P :

$$N:P\{p\} = N \cup P\{p\}.$$

where $P\{p\} = P \cup \text{translate}(P, p) \cup \text{translate}(P, 2p) \cup \dots$. A periodic element is essentially an infinite union of intervals with the periodic part repeating indefinitely as shown in Figure 2.

If the periodic part of a periodic element is an empty element ($P = \emptyset$) or the period is zero ($p = 0$), then this periodic element reduces to a temporal element and the periodic part is omitted in representation and written N

for short. This means that existing aperiodic temporal database models can be extended with periodic elements. If the aperiodic part is an empty element ($N = \emptyset$), then it is omitted from the representation and written $P\{p\}$ for short. Functions *period*, *periodic_part*, and *aperiodic_part* return the period, aperiodic part and periodic part of a given periodic element respectively:

$$\begin{aligned} \text{period}(N:P\{p\}) &= p, \\ \text{aperiodic_part}(N:P\{p\}) &= N, \\ \text{periodic_part}(N:P\{p\}) &= P. \end{aligned}$$

In order to accommodate Boolean connectives *and*, *or*, *not* in a periodic temporal query language, we define the usual set theoretic operators for periodic elements. Periodic events are closed under set theoretic operations of *union* \cup , *intersection* \cap , *difference* \setminus , and *complementation* $\bar{}$ [11]. These operations are depicted in Figure 3. Formal semantics are omitted because space limitations.

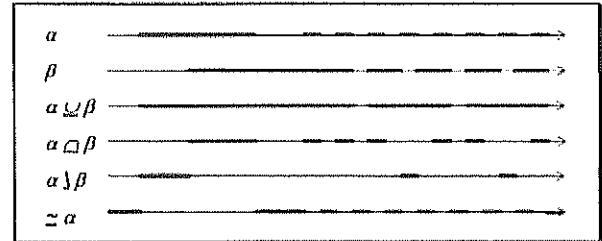


Figure 3: The set theoretic operators on periodic elements.

A temporal event is a happening whose temporal aspect is of importance to the user. Temporal events can be categorized into 3 groups: (i) *aperiodic*, (ii) *strictly-periodic* and (iii) *partially-periodic* (or periodic for short) events as shown in Figure 4 (Finite periodic events are described later).

Definition 2: A temporal event represented by a periodic element $N:P\{p\}$ is called a *partially-periodic event*. A *partially-periodic event* $N:P\{p\}$ is called a

¹ *before* is a temporal relational operator defined as follows: $[a, b]$ before $[c, d] = \text{true}$ iff $b < c$.

² Functions *last_instant* and *first_instant* return the beginning and end points of a given interval respectively.

strictly-periodic event and denoted by $P\{p\}$ if the aperiodic part is empty, i.e., $N = \emptyset$. A strictly-periodic $P\{p\}$ event is called an aperiodic event denoted by P if its period is zero, i.e., $p = 0$.



Figure 4: Temporal Events

Example 2: A periodic element $(I_1, I_2, \dots, I_n) : (J_1, J_2, \dots, J_m)\{p\}$ can be used to represent different type of events where I s and J s are intervals in the aperiodic and periodic part of the event. The employment period of a full-time employee who started on Feb. 3, 1970 and retired on Mar 5, 1994 is an aperiodic event represented by the periodic element $[70/2/3, 94/3/5]$ which is an interval: The work hours of an employee (Rick) working part time between 8:00am-12:00pm every morning is a strictly periodic event shown on the first row in Table 1. The constant day represents a duration of length one day. We use a time stamp of the form year/month/day/hour/minute/second to represent time instants. If the same employee starts working in the afternoons every other day on Jan 16, 1994, the work hours of the employee becomes a partially-periodic event as shown on the second row. If this employee quits on May 5, 1994, his work hours becomes an aperiodic event. We employ a construct called finite periodic element to represent a long repetitive temporal element. For a given strictly-periodic event $N\{p\}$ and an end time t (represented by an instant), the finite periodic element $N\{p\}^t$ is given as $N\{p\}^t = N\{p\} \cup [0, t]$. Finite periodic elements are formally treated just like a temporal element and are used for a compact representation. They can be used like an interval in the representation of periodic elements (the second and the third rows), i.e., a periodic element can be written as $(a_1, a_2, \dots, a_n) : (b_1, b_2, \dots, b_m)\{p\}$ where a s and b s are finite periodic elements or intervals. They provide an efficient storage of temporal values. Events represented by finite periodic elements are called finite periodic events (Figure 4).

Table 1: The evolution of a periodic event.

Changes	Temporal Event
On 94/1/1 working from 8:00 to 12:00 every day.	$[94/1/1/8/00, 94/1/1/12/00] \{1\text{day}\}$
On 94/2/1 working from 8:00 to 12:00 every other day.	$[94/1/1/8/00, 94/1/1/12/00] \{1\text{day}\}_{94/1/16} : [94/1/16/13/00, 94/1/16/17/00] \{2\text{days}\}$

On 94/5/3 Rick quits.	$[94/1/1/8/00, 94/1/1/12/00] \{1\text{day}\}_{94/1/16}, [94/1/16/13/00, 94/1/16/17/00] \{2\text{days}\}_{94/5/3}$
-----------------------	---

3. Periodic Temporal Object-Oriented SQL

Being a standard query language, SQL-92 is recently extended for temporal data, called TSQL2 [17], a language specification intended to be a consensus temporal extension. Though TSQL2 or SQL-92 itself can be extended with periodic elements to model periodic events, we choose SQL3 to illustrate the suitability of periodic elements, because it is an object-oriented language with abstract data types and is a superset of SQL-92 with some exceptions [10]. This allows us to define temporal values as a function of time without increasing the complexity of the language. SQL3 is currently being revised periodically and is intended to be a standard in the 1995-1996 time frame [Ga 94, Me 94]. The aperiodic part of our temporal extension is similar to that of Dayal and Wu [4] where time is represented as a function from temporal domain into attribute domains. We discuss the SQL3 proposal briefly in this section. The periodic temporal extension to SQL3 is discussed in the next section.

3.1 SQL3

SQL3 has all the fundamental object-oriented concepts which can aid in the development of a temporal extension. Every object is assigned an immutable object identifier in SQL3. An object identifier uniquely identifies an object and never changes. Objects and classes are defined through abstract data types (ADT) which can be constructed from the base types or existing ADTs through construct like tuple, set, multi-set, list. Attributes are defined with an encapsulation level of *public*, *private* or *protected*. Methods and attributes that are public constitute the interface of the ADT and are visible to other ADTs. Components that are private are visible only within the definition of ADT and can not be referenced outside the ADT. Components that are protected are visible only in the subtypes of the ADT and within the ADT itself.

A new ADT can be defined as a subtype of an existing ADT. A type can have more than one subtype and more than one supertype. An instance of a subtype is also an instance of its supertypes. Every instance is associated with a most specific type corresponding to the lowest type assigned to the instance. An instance

can have exactly one most specific type. Inheritance provides an abstraction mechanism by relating the classes of objects hierarchically. The methods and attributes of a supertype are inherited by its subtypes. A type can have more than one direct super type (multiple inheritance). SQL3 supports polymorphic functions and parameterized types. Supported type constructs include *set*, *multi-set*, and *list*. A set is a collection of objects without duplicates. A multi-set is a collection of objects with duplicates. A list is a multi-set with order among its members.

Since SQL is a table based language, the objects in SQL3 exist within a table. SQL3 allows a specification of a table over an ADT class. A table is considered as a set of objects of an ADT. CREATE TABLE T OF <ADT name> creates a tabular envelope around the ADT type specified by <ADT name>. The attributes of the ADT become the columns of table T. Each tuple of table T holds an instance of ADT <ADT name>. The objects are managed by the usual SQL insert, update, delete statements.

3.2 Periodic Temporal Data Model

Temporal events can be modeled by associating time with either objects or by attributes called *attribute stamping* and *tuple stamping* respectively. Tuple stamping is achieved by introducing one or more columns in a table to represent the valid time of tuples in relational model. Attribute stamping associates time-stamps with each value, thus resulting in a non-first normal form relation. In an object oriented model, time can be associated with an attribute or with the object itself. Associating time with an object causes all attributes to be copied in the history of the object, when the value of only one attribute is changed. We associate time with attributes of an object which may be objects themselves through the *part-of* relationships. The valid time of a complex object is recursively derived from its constituent objects.

The history of an attribute is stored as a function of time from the attribute domain into the time domain (The meaning of the term *history* extends into the future in the case of periodic elements). This function is stored as a set of mappings called *temporal mapping* such as $Salary = ([89/1/1, 95/5/1] \rightarrow \$2000, [95/6/1, 96/7/1] \rightarrow \$2100)$ for a salary attribute. *Lifespan* of an attribute or object is the time over which it exists or happens. *Vtime* function computes the lifespan of an object/value by unioning the periodic elements in the temporal mapping of the object/attribute. The lifespan is $Vtime(Salary) = [89/1/1, 96/7/1]$ for the salary attribute. The lifespan of a complex object is the union of the lifespans of its constituent objects and attributes

defined recursively. The lifespan of a tuple is the union of the lifespans of the attributes/objects in the tuple. The lifespan of a set/list/multi-set of attributes/objects is the union of the lifespans of the attributes/objects in the structure. Since a class is a set of objects, the lifespan of a class is derived from the union of the lifespans of the objects in the class. The lifespan of a superclass is the union of the lifespans of its subclasses.

3.3 Temporal Type Hierarchy

We depend on the advanced object-oriented ADT capabilities of SQL3 to model time. By making time an ADT, the implementation is hidden from the user. We introduce periodic time into SQL3 by defining a type hierarchy of temporal types shown in Figure 5. A simplified version of the type definitions are given in Appendix in a SQL3-like syntax. Different semantics of time such as discrete, continuous, bounded, unbounded, etc. can be defined by user to suite specific needs of an application by fully/partially overriding the built-in temporal ADTs defined here. The type hierarchy consists of two set of types: *absolute temporal types* and *relative temporal types*. The absolute temporal types are *instant*, *interval*, *element* (temporal element), and *periodic_element* in the order of increasing complexity. The relative temporal types are *relative_interval*, *relative_element*, *relative_periodic_element* which are relative counterparts of absolute temporal types. Relative temporal types represent relative events with an unspecified beginning time.

Figure 5: Simplified Temporal Type Hierarchy.

Periodic elements represent relative events by marking the beginning instant of the event with * symbol. For example, the relative temporal element $([*e, 5], [10, 15], [20, 25])$ denotes an event happening 3 times for 5 time units with 5 time units intervals in between. The relative periodic event $[*e, 2]{5}$ represent an event happening for 2 time units in every 5 time units from the unknown beginning. The *relative periodic element* type can be used to represent temporal *templates* such as predefined work schedules, future plans of activities with no specific times, presentation of multimedia objects, or a sequence of scientific experiments with relative timings. We won't elaborate on relative events because of space limitations. It suffices to say that they are more or less treated in the same way as absolute events with most of the operators on absolute events applying to them in a similar fashion. For example, temporal

scaling can apply to a relative in the same way it applies to an absolute events. However temporal translation does not make sense for relative events. These argument can be carried over to set theoretic operations (union, intersection, complementation) and relational operators (contains, equal, starts, ends overlaps etc.) as well.

The instances of temporal types are accessed and modified through operators. There are four types of temporal operators associated with temporal types:

1. Specialized Operators: Functions defined specifically for a temporal type such as `first_interval`, `last_interval`, `number_of_intervals` etc. on the type the element and `relative_element` types.
2. Relational Operators: Operators for capturing the

the behavior of objects. *ref (T)* is a reference type for type T. *function (T₁) returns T₂ end function* is a function mapping from type T₁ to type T₂. The employee data is stored in *employee* class defined below:

```

create type employee
(
  employee#    virtual,
  salary       virtual,
  department   virtual,
  function employee# (:e ref(employee))
                returns integer
end function,
function salary (:e ref(employee)) returns
  function (periodic_element) returns integer
end function
end function,

```

Table 2: The Employee class/relation with periodic elements.

Name	Salary	Task
[91/1/1, now] → Mary	[91/1/1, now] → \$1000	[91/1/1, now] → Toy
[91/1/1, 97/1/1] → Jack	[91/1/1, 95/1/1] → \$2000 [95/2/1, 97/1/1] → \$2300	[91/1/1, 94/3/1] → Shoe [94/4/1, 97/1/1] → Toy
[92/1/1, ∞) → John	[92/1/1, 92/6/1]{1year} → \$1100 [92/7/1, 92/12/1]{1year} → \$1200	[92/1/1, 92/3/1]{6months} → Toy [92/4/1, 92/6/1]{6months} → Shoe
[94/1/1, ∞) → Tom	[94/1/1, ∞) → \$5	[94/1/1/8/30, 94/1/1/12/30]{1day} → Toy
[93/6/31, ∞) → Tim	[93/6/31/13, 93/6/31/17]{2days} → \$6 [93/7/1/13, 93/7/1/17]{2days} → \$4	[93/6/31/13, 93/6/31/17]{2days} → Shoe [93/6/31/8, 93/6/31/12]{2days} → Toy

temporal relationship among events such as before, after, starts, contains etc.

3. Temporal Transformations: Translation and scaling functions for expressing temporal changes.
4. Set theoretic operators: These operators are used to express the complex events in terms of simples once through Boolean connectives and, or and not.

3.4 Queries in Periodic Temporal SQL3

We express temporal queries in a SQL3-like syntax. Consider the employee database described in Section 1 which contains both aperiodic data (schedule data for full-time employees) and periodic data (schedule data for part-time employees). There are five employees as shown in Table 2. Three of these employees (Mary, Jack, John) are full-time with the rest being part-time (Tom, and Tim). Note that John rotates between *Toy* and *Shoe* department in every 3 months and he is paid \$1100 and \$1200 a month for the first and the second half of the year respectively. We use the virtual functions of SQL3 to uniformly model attributes and

```

function department (:e ref(employee)) returns
  function (periodic_element) returns
    ref(department)
end function
end function,

```

)
 create class employees of employee. Non-temporal attributes like *employee#* are defined as a function returning the value of the attribute. *employee#(:e)* returns the employee number of object *e*. Temporal attributes like *salary* and *department* return a function (the temporal values are a function of time) instead of a value. The attribute value at a specific instant can be retrieved by invoking this function with a temporal value. For example, *salary(:e)* returns a function which is can be used as *salary(:e)(94/12/31)* returning the salary on Dec. 31, 1994.

Query 1: What was Tim's salary when he first started, now, and in Dec. 1998.

```

Select  salary(e)(first_instant(t)),
        salary(e)(now), salary(e)(98/12/1)
from    employees e, Vtime(salary(e)) t

```

where name(e) = 'Tim'

t is a shorthand for $Vtime(salary(e))$ which returns the lifespan of the salary attribute of object identified by e . The overloaded function *first_instant* returns the first instant of a given periodic element. Since salary value is a function of time, the salary on a particular instant is obtained by invoking the *salary* function $salary(e)$ of object e with a time parameter. When the time parameter is omitted it is assumed to be *now*. If time parameter is a periodic element, the values of attributes falling into the time given by the periodic element are returned as a set of temporal mappings.

Query 2: Status of employees may change between full-time and part-time status over time. List current part-time employees assuming part-time employees work on periodic schedules.

```
Select name
from employees e
where period (Vtime(department(e))) ≠ 0
```

Part-time employees can easily be identified by checking if the period of their work-hours is zero (aperiodic event) or not (periodic events). Full-time employees can be determined from the period or the periodic part of the schedule similarly. Function *period* is defined in Section 2.

Query 3: Assuming Tim will always be working the same number of hours, how many hours will he have worked by year 2000? How many times a week he comes to work currently?

```
Select length(t ⊆ [now, 99/12/31]),
number_of_intervals(t ⊆ [95/4/17,
translate(95/4/17, 1week)])
from employees e, Vtime(department(e)) t
where name(e) = 'Tim'
```

Function *length* returns the sum of the interval lengths in a periodic element. $t ⊆ [now, 1999/12/31]$ is Tim's work hours between the beginning *now* and year 2000. Function *number_of_intervals* counts the number of intervals in a temporal element. $[95/4/17, translate(95/4/17, 1week)]$ returns an interval of length 1 week in the future assuming *now* is earlier than Monday 95/4/17. Function *translate*(α, d) takes a periodic element α and shifts it by d time units.

Query 4: When an employee goes on vacation, his work is to be transferred to another employee. Who are those employees who can substitute for Tim?

```
Select name(e)
from employees e
where name(e) ≠ 'Tim' and
Vtime(department(e)) contains
Vtime(Select Vtime(department(f)) ⊆ [now, ∞)
from employees f
where name(f) = 'Tim')
```

The subquery above returns a set of periodic elements. Remember that the lifespan of a set of objects/values is the union of the lifespans of the objects/values in the set. Therefore an implicit union is performed by the *Vtime* function on the set of periodic elements. $Vtime(subquery)$ computes Tim's schedule after *now*. Note that expression $Vtime(department(f))$ returns the whole schedule over the future and the past, while the intersection $Vtime(department(f)) ⊆ [now, ∞)$ returning only the future part of the schedule. Operator *contains* checks if a temporal event contains another, written infix format for brevity. It is a direct extension of *contains* operator on intervals. We illustrate some of these relational operators on periodic elements in Figure 6.

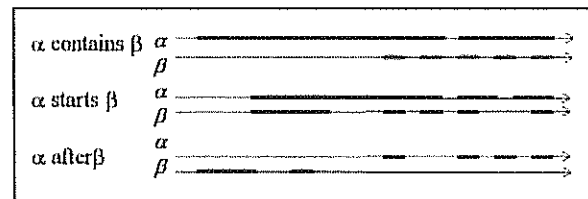


Figure 6: Relational operators on Periodic elements

Query 5: Compute the time during which no part-time employee works after Jan 2, 1989 assuming part-time employees work on either daily or weekly basis?

```
[89/1/2, ∞) \ Vtime (Select Vtime(t)
from employees e
where period(department(e))=1 day or
period(department(e))=1week)
```

Interval $[89/1/2, ∞)$ is the time after Jan. 2, 1989. Difference operator $\alpha \setminus \beta$ computes the time in which event α occurs and event β does not occur.

Query 6: What is the current number of employees?

```
Select count(e)
from employees e
```

This is a nontemporal query counting the number of employee at instant *now*.

Query 7: What is the number of employees over the history of the company?

```
Select count(objects(e)(t))
from employees e, Vtime(employees) †
```

where $objects(e)(t)$ returns the set of objects of type e over time t .

4. Optimization of Periodic Expressions

Optimization is important for efficient query processing. Optimization of periodic expressions

involving periodic elements and operators can be a part of global query optimization. Periodic expressions are expressions constructed through the relational comparison operators, set theoretic operators, special operators and transformation functions mentioned in Section 3.3. Since periodic elements are capable of representing aperiodic, periodic, absolute and relative events in a straightforward manner, different types of events may be mixed and optimized. Set theoretic operators on periodic elements constitutes a Boolean algebra with a set of optimization rules such as associativity, distributive law. Expressions with transformation and temporal comparison operators can be optimized in a similar manner [11].

Over the course of time, historical data accumulates in large amounts. Therefore representation and storage of temporal data is an important factor in the data model. Periodic element can represent aperiodic and periodic data efficiently in *canonical form*. We define a canonical form for periodic elements to reduce redundancy in representation by removing the repetitive data. Consider the periodic element below.

$[0, 2], [2, 5], [10, 12], [15, 17] : [20, 22], [25, 26], [26, 27]\{10\}$

which can be represented more efficiently by the following periodic element $[0, 5] : [10, 12]\{10\}$. Even more efficient representation can be obtained representing finite periodic events using *finite periodic elements* defined in Section 2.

5. Related Research

A framework for handling possibly periodic time based on constraints in relational data model is proposed in [12]. A temporal event is represented by what is called a *generalized tuple* consisting of a set of temporal attributes, a set of time stamps, and a set of constraints. Two time stamps are used to represent possibly infinite set of intervals. Each time stamp is represented by a *linear repeating point (lrp)*. A *lrp* is the set of time instants computed by the formula $a+kn$ where a and k are integer constants and n takes integer values between $-\infty$ and ∞ . For example, $2+3n$ corresponds to the following set of instants $\{\dots, -4, -1, 2, 5, 8, \dots\}$. Then two *lrps*, one for beginning time chosen from the first *lrp* and the other for end time chosen from the second *lrp*, represent an infinite union of the intervals satisfying the given set of constraints. Consider the following generalized tuple from [12]:

$$[3+2n_1, 5+2n_2], X_1=X_2-2$$

where $X_1 = 3+2n_1$ and $X_2 = 5+2n_2$ representing the following infinite set of intervals: $\{\dots, [1, 3], [3, 5],$

$[5, 7], \dots\}$ which actually corresponds to the interval $(-\infty, \infty)$.

Generalized tuples are defined for temporal relational model using tuple stamping. It is not clear how this approach can be applied to attribute stamping. Generalized tuples are biased towards constraints programming. The constraints and operations on generalized tuples modify the relational data model dramatically in order to incorporate time into the model resulting in exponential time query evaluation. We believe that using periodic elements to represent periodic events in temporal databases is a simpler and more natural alternative to using generalized tuples.

There are studies modeling for calendars which are inherently periodic entities. Among these are collection of intervals [Le 86, Ca 94] and the SQL2 extension for multiple calendar support [18]. Temporal mediators [WaLa 93] based on intervals are used for displaying data in temporal databases in different time units. All of these models can represent aperiodic events. The SQL2 extension and temporal mediators support only aperiodic events. Collection of intervals can represent aperiodic and strictly-periodic events but not partially-periodic events. In our model, periodic elements makes a clear distinction between the aperiodic and strictly-periodic parts of a partially-periodic event. Set theoretic operators are supported between periodic events in our model. the SQL2 extension and temporal mediators support set theoretic operators only for aperiodic events. There is no set theoretic operator in collection of intervals.

6. Conclusion

Periodic temporal events occur very frequently in real life. Overwhelming majority of temporal data models deal with aperiodic events. Existing temporal types of instant, interval and element can be used model aperiodic events but not periodic ones. A periodic type has to conform to some requirements such as simplicity of representation, ability to represent aperiodic, strictly-periodic and partially-periodic, relative, absolute events uniformly. We proposed a temporal type called periodic element meeting these requirements and showed its applicability to temporal databases by extending object-oriented SQL for temporal data. Our extension defines time as a set of abstract data types complete with special operators, relational comparison operators, set theoretic operators and temporal transformations. These operators may be overridden by user to define different semantics of time. Temporal expressions involving periodic elements can be optimized for efficient query processing. Since instants, intervals and temporal elements are periodic elements by definition,

existing temporal models and their query languages can be extended for periodic data without changing the syntax and semantics. Periodic temporal data can be stored efficiently using canonical form to reduce the amount of data stored in the database.

Periodic events have not been studied extensively in the context of temporal databases. Many issues concerning the periodic temporal databases need to be studied such as temporal joins, query optimization, temporal indexes. Periodic elements can also be used to model different types of domains such as active databases, real-time systems, multimedia streams uniformly. A study about modeling calendar with periodic element is currently being conducted [11]. The rest remains as future work.

References

- [1] Ilsoo, A., *SQL+T: A Temporal Query Language*, Proceedings of the ARPA/NFS International Workshop on an Infrastructure for Temporal Databases, Arlington, Texas, June 14-16, 1993.
- [2] Allen, J. F., *Maintaining Knowledge About Temporal Intervals*, Communications of the ACM, 26(11):832-843, November 1983.
- [3] Chandra, R., et al., *Implementing Calendars and Temporal Rules in Next Generation Databases*, Proceedings of the Third International Conference on Data Engineering, IEEE 1994.
- [4] Dayal, U., and G. T. J. Wu, *Extending Existing DBMSs to Manage Temporal Data: An Object Oriented Approach*, Proceedings of the ARPA/NFS International Workshop on an Infrastructure for Temporal Databases, Arlington, Texas, June 14-16, 1993.
- [5] Gadia, S., *A Homogeneous Relational Model and Query Languages for Temporal Databases*. ACM Transactions on Database Systems, 13(4):418-448. December 1988.
- [6] Gallagher, L., *Influencing Database Language Standards*, Sigmod Record, Vol. 23, No. 1, March 1994.
- [7] Gadia, S., and B. Bhargava, *SQL-like Seamless Query of Temporal Data*, Proceedings of the ARPA/NFS International Workshop on an Infrastructure for Temporal Databases. Arlington Texas June 14-16, 1993.
- [8] Gadia, S., et al., *A Query Language for Homogeneous Temporal Data*. ACM-SIGMOD 1985.
- [9] Gadia, S., et al., *An SQL-like Seamless Query Language for Spatio-temporal Data*. Proceedings of the ARPA/NFS International Workshop on an Infrastructure for Temporal Databases, Arlington, Texas, June 14-16, 1993.
- [10] Kulkarni, K. G., *Object-orientation and the SQL Standard*, Computer Standards & Interfaces, 15(1993) pp. 287-300, North Holland, 1993
- [11] Kurt, A., *Modeling Periodic Temporal Databases with Periodic Elements*, Technical Report (in preparation), Case Western Reserve University, Cleveland, Ohio, 1995.
- [12] Kabanza, F., *Handling Infinite Temporal Data*, Ninth Annual ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 392-403, Nashville, TN, April 1990.
- [13] Leban, B., McDonald, D. D., and D. R. Forster, *A Representation for Collections of Temporal Intervals*, AAAI pp. 367-371, 1986.
- [14] Melton, J., *ISO/ANSI Working Draft SQL (SQL3)*, August 29, 1994.
- [15] Rose, E., and A. Segev, *TOODM - A temporal Object-oriented Data Model with Temporal Constraints*, Proceedings of the 10th International Conference on the Entity Relationship Approach, October 1991.
- [16] Segev, A., and A. Shoshani, *Logical Modeling of Temporal Data*. ACM SIGMOD 1987.
- [17] Snodgrass, R., et al., *TSQL2 Language Specification*, Sigmod Record, 23(1), pp. 65-86, March 1994.
- [18] Soo, M., *Multiple Calendar Support for Conventional Database Management Systems*, Proceedings of the ARPA/NFS International Workshop on an

- Infrastructure for Temporal Databases*. Arlington Texas June 14-16, 1993.
- [19] Su, S. Y. W., and H. M. Chen, *A temporal Knowledge Representation Model OSQM*/T and its Query Language OQL/T*. *Proceedings of the Conference on Very Large Databases*, Barcelona, Spain, December 1991.
- [20] Tansel, A., *Adding time Dimension to Relational Model and Extending relational Algebra*, *Information Systems*, 11(4), 1986.
- [21] Tansel, A., *A Historical Database*, *Information Sciences*, No 53, pp. 101-133, 1991.
- [22] Tansel, A., et al., *Temporal Databases: Theory, Design and Implementation*. Benjamin/Cummings Publishing Company, Inc., series on databases systems and applications (book), 1993.
- [23] Wang, X. S., and S. Jajodia, *Temporal Mediators as a way to Support Multiple Temporal Representations*, *Proceedings of the ARPA/NFS International Workshop on an Infrastructure for Temporal Databases*, Arlington, Texas, June 14-16, 1993.
- [24] Wu, G., and U. Dayal, *A Uniform Model for Temporal Object-oriented Databases*. *Proceedings of the International Conference on Data Engineering*, Arizona, pp. 584-593, February 1992.

Appendix

```

create type instant under interval, ( . . . )
create type interval under element, ( . . . )
create type element under periodic_element,
(
  . . .
  /***** Special Functions *****/
function next_interval (element, interval) returns
interval
end function,
function previous_interval (element, interval) returns
interval
end function,
function interval (element, integer) returns interval
end function,
function length (element) returns relative_interval
end function,
function #of_intervals (element) returns integer
end function,
function distance (element, element) returns
relative_interval
end function,
function closest (element, instant) returns interval
end function, /* Returns the closest interval to the
given instant in the element. */
  /***** Temporal Transformations *****/
function translate (element, relative_interval) returns
element
end function,
function scale (element, relative_interval) returns
element
end function,
  /***** Relational operators *****/
function before (element, element) returns boolean
end function,

```

```

/* Other relational operators are defined similarly */
  /***** Set theoretic operators *****/
function union (element, element) returns element
end function,
/* Other set operators are defined similarly */
)

create type periodic_element
(
  aperiodic_part virtual,
  periodic_part virtual,
  period virtual,
  /***** Formatting functions, used to display data in
different ways *****/
function left_shift (periodic_element, relative_interval)
returns periodic_element
end function,
function right_shift (periodic_element,
relative_interval) returns periodic_element
end function,
function fold (periodic_element, relative_interval)
returns periodic_element
end function,
function unfold (periodic_element, relative_interval)
returns periodic_element
end function,
  /***** Set theoretic operators *****/
function union (periodic_element, periodic_element)
returns periodic_element
end function,
/* Other set theoretic operators are defined similarly.
*/
  /***** Relational Operators *****/
function contains (periodic_element,
periodic_element) returns periodic_element
end function,
/* Other relational operators are defined similarly */
  /***** Temporal Transformations *****/
function translate (periodic_element, relative_interval)
returns periodic_element
end function,
function scale (periodic_element, relative_interval)
returns periodic_element
end function,

```