UNIVERSITY OF OSLO
Department of Informatics

Temporal Data: Time and Relational Databases

Bjørn Skjellaug

Research Report 246

ISBN 82-7368-161-0
ISSN 0806-3036

April 1997

Temporal Data: Time and Object Databases

Bjørn Skjellaug[1]
Institutt for informatikk
Universitetet i Oslo
P.O.Box 1080 Blindern, 0316 Oslo, Norway

bjornsk@ifi.uio.no
http://www.ifi.uio.no/~bjornsk/

Research Report 246
ISBN 82-7368-161-0

April 1997

[1]The author is also part time at Department of Distributed Information Systems, SINTEF Telecom and Informatics, Oslo.

Contents

Chapter 1

Introduction

> What, then, is time? I know well enough what it is, provided that nobody asks me; but if I am asked what it is and try to explain, I am baffled.
>
> Saint Augustine in the 'Confessions',
> Book XI, Section 14. [7, page 36].

Most computerized information systems are formed from general purpose components, such as databases, GUIs, etc. Whenever properties of, and operations on, these components are generic they are in principle reusable for different purposes and in different contexts. Time and time support may also be regarded as a generic information system component. In fact, most applications have requirements that involve dynamics, and a time-varying nature of both data and processes used by these applications. Temporal databases are means to capture some aspects of the required time support, and are mainly used to manage historic data, present data, and predictive data within the same framework and model. To be more specific, in our context time acts as an intrinsic part of objects denoting when these objects are defined related to the built-in time dimension(s) support of the system. A common name of such objects is temporal objects, i.e., objects capturing, besides other properties, time and time dependencies of their values, e.g. an address history of a person.

1.1 'What, then, is time?'

Time has been a topic of philosophical and natural scientific study since the ancient Greeks (including Plato, Aristotle, and Diodorus Cronus), through Saint Augustine in the Middle Ages and many later Medieval logicians, to Newtonian physics, Einstein's theory of relativity, and on to present [49]. In the second half of our century logicians, computer scientists, and others have

showed significant interest in the task of understanding time in both more formalized, linguistic, mathematical, and technical terms.

The material presented in this report is founded on the above results, in particular, and not surprisingly, mainly of those results utilized and developed by computer scientists.

Time is a human construct. Although it is useful for describing and prescribing changes to the systems and objects under study, as we will see later on, time is in the day to day life most often understood as something which is ever increasing, or is continuously moving in one direction – into the future. This perception of time is more (or only) reflecting human memory. That is, we are able of memorizing the past, e.g. as facts and experiences, but we know nothing about a given time in the future. Still, we may have expectations for and predictions about the future. But, it is only when we reach that time in the future, which in the past was a future time, we can say anything, for example, about the validness of our predictions at the time when the predictions were first stated.

Another model presented by Rucker [57] incorporates time and space into a 4-dimensional space where all "points" are predefined. In such a model we are not "moving" in any specific directions; humans and things are objects defined within the hyperspace. Hence, each 4D object is always showing us when and where we were, are, and (possibly) will be. So, both past, present and future exist simultaneously. Although some points may be regarded as "facts" and others are "predictions", we are within that hyperspace all the time – "going" nowhere – because all entities are defined by their extent in this 4D-space. Rucker elaborates on this 4D view most elegantly in his book: "Geometry, relativity and the fourth dimension" [57].

1.2 Time and databases

In this report we are aiming at an understanding of time as an integral part of properties of data objects. Basically this reduces to understand how properties (or objects) change and behave over time. A common name of such objects is temporal objects, i.e., objects denoting, besides other properties, time and time dependencies. There are different "times", and different notions of time for data management may denote whether phenomena in reality are regarded as *events*, *facts*, or *processes*.

An intuitive interpretation of time objects in a system would be to let events be represented by time points, facts by time periods (or intervals) and processes with time functions. Why these interpretations? To follow this more or less common sense notion of time a bit further: An event is often taken as happening at a time instant, i.e., a point in time with no duration. A fact is a truth about some static aspects of reality but does not last for ever, i.e. a valid-time period defining when the fact was, is, or is

believed to be defined (i.e., valid) in reality. A process like an event denotes or refers to a dynamic aspect of reality, and which, unlike an event, does have a duration. Thus, a process may be a function describing some "cyclic" or repeating behavior like a tide or a continuous behavior like an expanding desert or disease. A process and a fact may for example be defined to last forever, or until terminated or changed, respectively.

We will primarily treat time related to facts because our concern is temporal data and how such data could be used to integrate facts of different kind and/or of different databases. We easily see that if an object changes its value(s) this is in the first place caused by an external/real event triggering some action to do the change (e.g. database update operation). The event itself may have been initiated by a process object. So in a overall setting objects of facts, events and processes are inter-linked. We deal only with the former of these objects, i.e., data management of facts, but will treat the other two whenever they relate to issues discussed.

We also present time and temporal data in a broader sense than is usually the case with temporal databases. We define the notions of schema versioning/evolution, object versioning and configuration management to be part of a broad scope of temporal data. Definitions and examples of these notions are given and we show how they are related to times such as valid times and transaction times, respectively.

For definitions of concepts and glossaries related to temporal databases see Jensen et al. [31].

1.3 About this Report

This report is one out of two reports describing the approaches, proposals and other issues within the field of temporal database research. This report deals with temporal relational databases, and the other deals with temporal object-oriented databases [61]. The reports give a survey of their respective subfields, and they classify the proposed models according to structural characteristics and present representatives of every class in more detail. However, equally important is that the fundamental concepts defined by the field is presented, therefore, this introduction chapter and the next chapter are included in both reports to give all potential readers an introduction and hopefully a sound understanding of the fundamental temporal concepts as defined and utilized within the temporal database field.

After fifteen years of active research in the field of temporal databases, about 800 papers published, the first international workshop was held in Arlington, Texas, 1993 [62, 51]. This workshop showed the diversity of issues and topics covered by the field, and the same tendency is documented by the follow-up workshop held in Zurich, Switzerland, 1995 [16, 60]. We are not aiming at covering all the research related to temporal database, and will,

of course, only refer to those research publications that are relevant for our purpose. The reader who are interested in temporal database bibliographies is referred to Tsotras and Kumar's bibliography [70], and also the electronic bibliography, authored by Kline, containing nearly 1100 entries as of January 1996 [35].

1.3.1 Report Structure

In Chapter 2 we presents some basic concepts that are generic to most temporal database models. Chapter 3 presents a survey of temporal relational databases, i.e. models and languages, with a classification of the different approaches presented in the literature. We end this report with a short summary in Chapter 4.

1.3.2 Acknowledgments

I would like to thank Amela Karahasanovic, Ragnar Normann and Dag Sjøberg for comments and suggestions to a previous version of this report.

This research was supported by the Norwegian Research Council through grant MOI.31297.

Chapter 2

Temporal Data — Basics

In this chapter we introduce some generic properties and definitions which are relevant in most settings dealing with temporal data.

2.1 Perception and Construction of Time

A philosophical view of time will probably pin point that an event is not (or necessarily not) an instant of or point in time, and could never be an instant of time because an event happens and something that happens has to have some sort of duration. Otherwise, it would not happen. In natural languages the term event is (in an informal sense) a homonym because it may denote the implicit semantics of time differently: 1) *'the concert was quite an event'*, e.g. meaning an experienced duration of joy and pleasure, and 2) *'at the event he sat down he had a stroke'*, e.g. meaning that something happened at a sudden (or an instant).

So, what is an instant of or point in time? In life and thought we often regard time as continuous say like a river. Thus, a point in time is non-existing or is not an appropriate notion if we want to grasp what continuous time is. However, a point in time is a (mathematical) construct and a notion that gives a sufficient structure of time when it comes to formal reasoning, data management, etc.

With proper models and operations we may sufficiently approximate or describe phenomena by computer systems by time points or other constructs derived from time points, e.g. intervals and periods. Euclidean geometry does the same with natural spatial primitives like for example an extended body that could be approximated by means of theoretical notions like points and lines. That is, the notion of extended body is described in terms of points and lines. See van Benthem [7, chapter I.1] for a more in depth presentation of the above problems, including a discussion of "The Fleeting Now".

2.2 Some basic definitions

Before presenting the different subjects and issues raised and discussed we will clarify some concepts which are used throughout the text (and without any further explanations if not the context defines them separately or relates them to more specific structures). The term *temporal data* means the concept where data or say an object is defined to have some time related information associated with it, e.g. a "built-in" timestamp[1]. For example, in a person object with an property attribute address, and whenever the person changes his/her address the new address value is timestamped. More important, though, is that the previous address values are still accessible, in the sense that an application (or user) may retrieve the whole or a part of the address history of a person. Thus, an address contains both the current and previous addresses (and possibly a future address), as well as the time information associated with each address value. Hence, by means of a temporal data model we may get an integrated knowledge of both where a person lived and when this was. The time (or periods) when a person has addresses is called the lifespan of the address object (see below). The relationship between facts, e.g. a person's addresses, and times is shown in Figure 2.1.

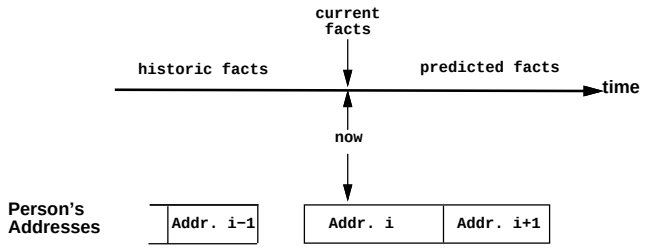

Figure 2.1: Facts and Times

The time information may be given different semantics depending on the time dimensions a particular system supports. For example, temporal database models usually define one or two of the following time dimensions: the *valid-time dimension* (see Section 2.3.1) supports information on when a fact was, is, or is believed to be valid (true) in reality, on the other hand the *transaction-time dimension* (see Section 2.3.2) supports information on when a fact was or is stored/current in the database. The transaction-time dimension has always its upper bound set to the (moving) time *now* and its lower bound is always set to the time when the system was first in operation, i.e., the time when the database was created. Whereas the

[1] A timestamp is a time value (point in time, time period, or time interval) associated with an object [31]. It is the key time representation in temporal databases. But, qualitatively different timestamps represent different notions of time. Hence, an object may have several orthogonal timestamps.

valid-time dimension's upper and lower bounds are set by the application, such bounds for the transaction-time are independent of applications. These two dimensions constitute an object's "real world" history and registration history respectively. In addition several database systems support primitive data types such as DATE, TIME, DATE-TIME, which are used to model what is called user-defined time, and its semantics is defined and only known by the application. For example, an object of type DATE is used to model the property of a person's date of birth, whereas a valid-time timestamp on that persons address property is used to handle the different times a person is associated with different addresses in the modeled reality. Thus, the built-in semantics of valid-time and transaction-time bound objects to the respective dimensions. That is, a time dimension provides more information of being only a value domain, i.e., a time related order of an objects values is maintained and this ordering is utilized, for example, by query processing.

Our concern is valid- and transaction-times and modeling their semantics as part of the database, and not only as structures and semantics of one particular application. Although valid and transaction-times are orthogonal temporal concepts there exist several applications where these two dimensions collapse into one dimension. Many (if not all) real-time database (RTDB) applications have this characteristic. For example, in a cash-line application a bank account is the temporal object of interest. A customer's withdrawal of an amount of money from his/her account means that the registration of this database transactions happens (approximately) at the same (instant of) time as the customers account is changed in reality, i.e. the time when the real money is withdrawn from the account and the time when this withdrawal is registered coincide, and only one (say valid-transaction-)time dimension may suffice for recording both times. Such an database is called degenerated if both valid- and transactions times are captured by one time value [32]. On the other hand when using an ordinary check the times when money is spent and when usage is registered, respectively, could vary form days to weeks, i.e., the valid-time (when spent) and transaction-time (when registered) do not coincide.

Further, a temporal data object is said to have a lifespan, and the time periods or intervals constituting the lifespan are not necessary contiguous. In the address example above, there could be a period when a person does not have a known address. Hence, the address object for this person was not defined for this period of time. Thus, the person had no address (or possibly an unknown address) for a period of time, e.g. see the "gap" between the address values `i-1` and `i` in Figure 2.1. So, the lifespan captures the time when an object is defined. That is, a valid-time lifespan is the time when the object is defined to exist in the modeled reality. On the other hand a transaction-time lifespan is the time when the object is defined to be current and accessible in the database [68, page 625].

We have referred to the concepts *point in time* and *instant* as synonyms.

Another important concept is *chronon*. A *chronon*, which is the shortest duration of time supported, for example, by a database is a non-decomposable unit of time or the smallest granule of time. Based on the notion of a chronon a time period is a set of (total) ordered time values, of which each is separated by a chronon. If a time dimension is discrete and linear the concepts of point in time, time period, and time interval are equivalent constructs for representing time [7]. Although this is true for the semantics of time, one construct may be preferred because applications interpret these semantics in context of the modeled reality. We return to this issue in a later chapter. In most temporal models the chosen time dimension is discrete [42, 50], and therefore it is isomorphic to (some subset of) the integers, or isomorphic to (some subset of) the natural numbers when it has an exact lower bound.

2.3 Time Dimensions and Definitions

Time dimensions are (usually) classified into valid-time and transaction-time dimensions. Both dimensions can be defined separately, and, hence, they are orthogonal.

2.3.1 Valid Time

The valid-time dimension supports not only management of histories and current data, but also planned or predicted data. Histories are data defined for previous times compared to the time *now*. Current data are data valid at the time *now* (i.e., present time), as opposed to predictive data which is data believed to be valid in the future. Figure 2.2 illustrates these properties of valid-time data. Thus, valid-time of some fact is the time when the fact was/is (believed to be) true in reality. Thus, a fact may have several instances, each with an associated timestamp recording changes of that fact [31].

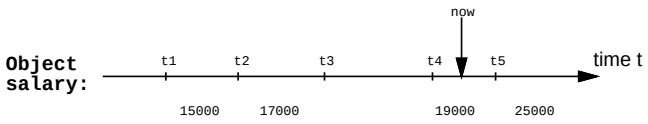

Figure 2.2: Valid-Time Data

Figure 2.2 shows an object salary. It could be an object recording the salary evolution of an employee of a company or a general salary object defining the salary of a particular professional category of a company, e.g. an assistant researcher, an engineer. Despite the difference in possible interpretation of the actual object say by an employee or a professional category application, the temporal semantics of this object remain the same. That is,

the object salary had from time t_1 to time t_2 a value of 15000, from time t_2 to time t_3 a value of 17000 and so on. Some interesting facts are also stored or are part of the semantics of the temporal object salary.

Firstly, between times t_3 and t_4 no value was recorded. We say that the object was not valid (or undefined) during that period. For the forementioned applications this could be interpreted as follows: 1) the actual employee in question had no salary (or even more likely s/he was not an employee) during that period, 2) the company had not a fixed salary for this professional category (or the professional categories did not exist) during that period.

The second interesting fact says something of what is planned or what is believed to be true (or be a fact) in reality, namely that the salary will be raised from 19000 to 25000 at time t_5 and will be fixed until some uncertain (unknown) time in the future. This could be interpreted by an application along the line of the employee and professional category examples above.

And last; the time *now* is in our example currently between times t_4 and t_5. We will discuss the issue concerning *now* and other variables in Section 2.3.4.

We have presented some of the goodies of a valid-time database, and given several examples of the semantics captured with this approach. A nice property with valid-time is that its basic semantics are independent of applications and their interpretations of the data. Thus, the basic temporal semantics are only managed by the built-in valid-time support of the database. This is true even when the valid times are usually given by the user. Hence, we distinguish on one side what valid-time is, and on the other for what it is used (or how it is used) by a user. The former concerns the database semantics the latter concerns some specific application's interpretation of it.

Below we discuss the transaction-time support where both semantics and usage are given and controlled by the database system. Valid- and transaction-times support increases the data independence of databases from applications. This aspect is the main technical argument for extending a database model to a temporal database model.

2.3.2 Transaction Time

The other time support in temporal databases is the so-called transaction-time. A transaction-time timestamp registers the time when an object (or say a fact) is current in the database, and may be retrieved [31]. As with valid-time a fact may have several value instances and timestamps associated with it. Although transaction-time never exceeds the current transaction-time (or time *now*) a fact has also associated with it transaction timestamps for predicted valid-time data when both time dimensions are supported by the database.

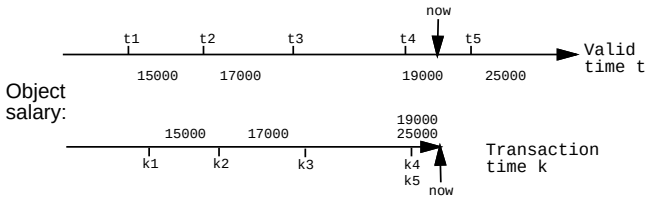

Figure 2.3: Transaction-Time Data

Figure 2.3 shows how the instances of the fact salary are associated with transaction timestamps. At time k_1 the salary value 15000 was first recorded in the database. And it was current in the database until transaction-time k_2. At transaction-time k_3 the fact was logically deleted but restored at transactions-time k_4 with the new value 19000. At the same time there is another transaction-time k_5 (equal to k_4) which records that the fact has value 25000, but not valid before t_5. Both values are current until the time *now*, which is always an upper-bound constraint on transaction timestamps. With transaction-time alone this information makes no sense besides saying something about when a particular value was stored. However, together with the valid-time support it will be interpreted as that both the current salary of 19000 and the planned/predictive salary of 25000 were recorded in the database within the same transaction or by concurrent transactions, respectively, but at the same time. This makes of course sense for both the employee and professional category applications above.

Transaction-times associated with facts have a property or restriction not shared with valid-times. That is, they are not to be altered when first stored. Put differently, we cannot change what already has happened, i.e., at previous transaction times. The transaction-times are in an one-to-one correspondence with the actual database transactions or operations performed, i.e., they register the modifications activities of the database.

2.3.3 Bi- and Multi-Temporal Dimensions

If a database captures both valid-time and transaction-time, we will have the relation shown in Table 2.1. This relation is the representation of the object salary shown in Figure 2.3. For convenience and hopefully to make the illustration more intuitive we think of this relation recording the salary of a professional category, and all symbolic times, t_i and k_j, are replaced with month-year timestamps, still, representing a similar information as above.

A database that supports both valid- and transaction-times is called a *Bi-temporal Database* [18]. In principle there are no limitations of how many time dimensions a database may support, and multi-temporal databases are in fact possible. In other words the 2D time associated with database facts

Salary	*Valid-Time*		*Trans-Time*	
15000	Jan88	Dec89	Apr88	uc^2
17000	Jan90	Mar92	Mar90	*uc*
19000	Sep94	Dec97	Oct94	*uc*
25000	Jan98	∞	Oct94	*uc*

Table 2.1: Object Salary represented as bi-temporal data

in Table 2.1 could easily be extended to capture n dimensions.

Jensen and Snodgrass present application scenarios of multi-temporal databases [32]. The multiple dimensions are based on a taxonomy where a valid timestamp is defined relatively to transaction timestamps. For example, a valid timestamp is restricted not only to one but several transaction timestamps, say when data is defined and stored both locally and globally by several databases. For one valid time there may be several transaction times and these times have restriction imposed on them. All such restrictions could be defined in a schema. A generalization of the multi-temporal nature of [32] is provided by STSQL, a spatio-temporal extension to SQL–92 [8]. In STSQL both multiple valid-times and transactions-times may be defined for a table. That is, multiple valid-time dimensions are associated with an object, where each dimension either denote a separate temporal aspect of an object or a possible world of an object. For example, if we have tasks to be planned and scheduled, one valid timestamp may denote the aspect of when the data stored about a task is true in reality, whereas another valid timestamp may denote the aspect of when the task itself should be executed, i.e., when it is scheduled. If we have possible world semantics, a third valid timestamp would possibly denote an other time when the task should be scheduled, because this time is estimated with some other parameters.

The value-added property of having bi- or multi-temporal relations is that these dimensions capture inter-dimensional semantics which are important for deducing information about the facts stored in the database. It requires, of course, that if such inter-dimensional semantics exist (i.e., their relationships are expressible), each dimension has to be identical, equivalent or compatible with some underlying notions of time. For example, the relation in Table 2.2 illustrates some of the relationships.

The second of Paul's rows, in Table 2.2, represents a so-called *pro-active* change of the database stating in December 1990 (start of trans-time) that Paul will be "re-hired" as an assistant from March 1991 (start of valid-time), i.e., the fact is stored before it is expected to be valid in reality. Paul's has a *retro-active* change of its position (row 4), i.e., in December 1991 the database registers that Paul was an engineer from August the same year. At this time it is also known that the information that Paul was an assistant until November 1991, in fact, was misleading information current

name	*department*	*position*	*Valid-Time*		*Trans-Time*	
Paul	design	assistant	Jan90	Sep90	Feb90	*uc*
Paul	design	assistant	Mar91	Nov91	Dec90	Dec91
Paul	design	assistant	Mar91	May91	Dec91	*uc*
Paul	design	engineer	Aug91	*uc*	Dec91	*uc*
Mary	design	manager	Jan90	Feb91	Jan90	*uc*
Mary	HQ	AD	Dec90	Dec93	Feb91	*uc*

Table 2.2: Bi-temporal Employee Data

in the database until December 1991 (row 3), i.e., Paul was an assistant only until May 1991. That is, all data retrieved before December 1991 may have resulted in inaccurate information. To restore a consistent database regarding Paul's position the following is done: The second row is logically deleted from the relation marked with a deletion time as the upper limit of the transaction timestamp. The information (as best known of December 1991) when Paul was an assistant this year is shown by the third row. The following row captures the present information that Paul is an engineer, and has been so since August 1991. Note that Paul was not an employee during the periods of October 1990 until February 1991, and in June and July of 1991, or Paul had no known position in the company during these two periods.

In the first of Mary's rows the two times coincide for her being a manager, i.e., the database and the reality modeled by it are synchronous in some sense on the fact that Mary was a manager during that period. The last row, together with the previous one, shows that during a specific period Mary held two positions in the company, i.e. from December 1990 until February 1991 Mary was both a manager and an AD.

There are a number of relationships between time dimensions such as those mentioned above. See Jensen and Snodgrass's discussion and taxonomy on temporal specialization and generalization for a more exhaustive presentation of this issue [32].

2.3.4 NOW and other variables

The time *now*, as defined by [15] and [24], in our example in Figure 2.2 is currently between times t_4 and t_5. A function with domain time could return the salary of the object at a particular time-value. In particular, the function *now* will always return the current salary of the object. Notice that the notion of *now*, and its like, makes temporal databases becoming variable databases. That is, we may store a salary say 10000 with a timestamp $[t_i, now]$ as a variable temporal object (not shown in Figure 2.2 and Table 2.1). However, *now* is a (continuously) moving target, so whenever *now* changes the timestamp changes and then the temporal interpretation

of the object changes. The same applies for queries: *"what is Mary's position now"*, which obviously would give different answers when *now* varies. In contrast, what is Mary's position as of May 1992 always yield the same answer.

Thus, it is a semantic difference between models supporting only fixed timestamp(s) and those who support variable timestamp(s). It could be thought of as a difference between extensional and intensional models as Clifford and Isakowitz point out, but without overloading these terms they named them variable databases [18].

2.3.4.1 Variables and Semantics

Alternatives and supplements for *now* are variables such as *uc* – until changed – [72], and distinguished individuals such as ∞[3] [63]. They all denote different semantics, but we will not discuss these issues here. However, it should be apparent that *now*, *uc*, and ∞ by their names denote different semantics.

Although such variables may be intuitively understood and used for the same purpose in different contexts/applications (and in fact they are used that way), we need a formal model to define them precisely. So, if the variable semantics managed by the database are unambiguous at the database level, the interpretation of what a variable denotes in a specific context is a transformation based on the database semantics but done by an application only.

For example, a personnel application and a project application would most presumably interpret the data of Table 2.2 differently because they operate in different contexts. A personnel "user" could interpret the data about Paul being an engineer (i.e. the variable *uc*) that he is still an employee. On the other hand a project "user" could interpret *uc* that he is a suitable candidate for a project. The semantics of *uc* are the same in both cases, but it gives rise to different information.

An interesting discussion on the issues concerning temporal variable databases and their semantics is found in [18].

2.3.4.2 Formal Issues on Variables

The function of *now* have been studied in several fields, for example in logic and philosophy. The pioneering work of A.N. Prior on time, modality and logic in the 1950's [53] and onwards [52] laid the ground for the formal study of *now*. Kamp gives a formal discussion and analysis of the English word *now* [34]. He studied *now*-calculi – $\mathcal{L}(\mathsf{N})$ – and their semantics in which *now* is an operator, and, hence, not a distinct individual. Not to be too technical, though, Kamp's main result is on completeness of the $\mathcal{L}(\mathsf{N})$'s axiom sets;

[3] Intuitively *forever* is the same as, and is also used as a synonym for, the distinguished individual ∞.

Such an axiom set is said to be semantically complete if a "same" calculus – $\mathcal{L}$ – without the *now*-operator has a semantically complete axiom set, and if $\mathcal{L}(\mathsf{N})$'s axiom set is "closely" related to $\mathcal{L}$'s axiom set. This is obtained easily in the propositional case, where every formula containing *now* in $\mathcal{L}(\mathsf{N})$ is proven equivalent to a formula not containing *now* in $\mathcal{L}$. For the predicate calculi the results are stated to be less general because (some) formulae of predicate calculi may not have any equivalent formulae without *now*. Hence, the completeness has to be proved by other means in this case.

Both Rescher and Urquhart [54, pages 35–37], and van Benthem [7, pages 6–7] have another interesting observation concerning the matter of *now* and temporal structures. The observation is by van Benthem stated as a "global property of temporal structures". The property is called 'homogeneity' and its definition imposes that all temporal individuals are formally indistinguishable. van Benthem requires no ontological difference between any time value t_i and for instance the time *now*. He argues that because the role of a/any *point in time* (e.g. *now*) can be played by any temporal individual, so when formalizing temporality the only interesting notions are those that are interesting in the general case. It seems to us that both Rescher and Urquhart, and van Benthem see in certain situations the necessity of a separate notion of for example *now*. Thus we interpret them such that distinguished individuals and their semantics have only interest in a context of an application domain, i.e., where a specific emphasis on such variables and distinguished individuals is required.

2.3.4.3 Databases — Application of Variables

We regard databases as a representative of an application domain of *now* and other distinguished individuals where it is crucial to define the semantics of such notions precisely. The term *individual* is somewhat loosely used here, what it really means is that it is a function value. The importance of a formal semantic is that a database has to respond unambiguously on the meaning of any access to (temporal) data. So, for *now* and other variables, the meaning of them should be uniformly and unambiguously for all access of the database. The database aspects of *now* are treated by Clifford et al. mostly relative to the temporal relational model [13]. The same authors have in particular discussed these issues in context of TSQL2 [14].

Chapter 3

Time and the Relational Model

3.1 Introduction

This chapter present the temporal relational model and its query language (QL) features. There has been an extensive research activity on relational temporal models and query languages over the past ten to fifteen years. The models and solutions proposed in the literature are based on adding time to the relational model, and extending the query languages, algebras and calculi.

Özsoyoğlu and Snodgrass present several published relational models and query languages, and their characteristics in a survey on temporal and real-time databases [50]. We will not present such models in full detail, but concentrate on the principles, i.e., issues and definitions, and give a representative examples of proposed models and their languages covering several approaches of incorporating time and temporal support.

3.2 The Conventional Relational Model

Codd's conventional relational model, [19], is a snapshot model, it doesn't take into account the time varying nature of reality, and it supports only one database state – the (assumed) current state, i.e., as best known now. Codd's model defines relations which are, in a mathematical and set-theoretic sense, defined as the Cartesian products over (value) domains, D_i. The mathematical notion of a relation is underlying Codd's notion of a relation. The more pragmatic concept of a relation is defined by a relation scheme R, which is a finite set of ordered attributes names, A_i, i.e., $R = (A_1, A_2, ..., A_n)$.

We can view a snapshot relation instance $r(R)$ (read as r is an instance of schema R) as a table where each row in r is an n-tuple value, $(a_1, ..., a_n)$. On the other hand each column corresponds to an attribute name, A_i, with an associated data type, e.g. an integer or string data type. A data type corresponds to the mathematical notion of a domain, and a column name,

A_i, in a table denotes a finite set of attributes values, a_i, of the associated data type.

Put differently, a tuple is a set of ordered name-value pairs (A_i, d_i), i.e., each element of the tuple maps an attribute name, A_i, onto its corresponding value, $d_i \in D_i$. Accordingly, a relation instance, $r(R)$, is defined as follows:

$$r(R) \subseteq \{(d_1, ..., d_n) | (d_1 \in D_1, ..., d_n \in D_n\}$$

The model defined by Codd is also the starting point for extending the relational model to a temporal model, where a relation scheme, $R = (A_1, A_2, ..., A_n|T)$[1], has associated a time representation T. The following presents several definitions and approaches to what the T is and how it is incorporated into the relational model.

3.3 Approaches to Support Time

In the literature several approaches define time support for different purposes, or they differ in their approaches on supporting logically the same purpose. The two main areas of time support for temporal relational databases focus on change management for data instances and change management for database schemata, respectively. We call these two foci for temporal extensional support and temporal intentional support, respectively.

3.3.1 Temporal Extensional Support

There are three main approaches where extensional time support is incorporated with the relational model. The first is to support some notions of user defined time (i.e., extend the number of simple data types with DATE, TIME, etc.).

The other two extend the relational model with DBMS built-in timestamp support and temporal semantics. One is timestamping tuples, and the other supports attribute timestamping. Thus, conceptually, they only differ on the level of timestamping.

Both of the timestamping approaches can have valid- and/or transaction-time support due to the orthogonality of these two time-dimensions. In the following we will mainly concentrate on tuple and attribute timestamping. The user-defined time approach is only given a very brief presentation in Section 3.4.

3.3.2 Temporal Intentional Support

The three approaches mentioned above only concern the temporal extensional aspects of data and not the temporal intentional ones, i.e., data instance versus schema issues, respectively. We first give a brief idea of the

[1] This notation of a generic temporal relational scheme is adopted from Jensen et al. [33].

latter. The temporal intentional issues, called schema versioning and/or schema evolution are often related to the transaction-time dimension where only the current schema is subject of change. That is, no previous schemata, and instances defined under them, are affected. This means that the registration history is not altered, only the current (i.e., the latest defined) schema and data may change properties. In Figure 3.1 a schema with the type Employee changes its definitions at times t_2 and t_4, respectively. These changes also affect the object named John (of type Employee) probably at the same or some time close to times t_2 and t_4, respectively. (For illustration only, the object John is also modified both at times t_3 and t_5, but, of course, these modifications do not affect the schema definitions of the object John.)

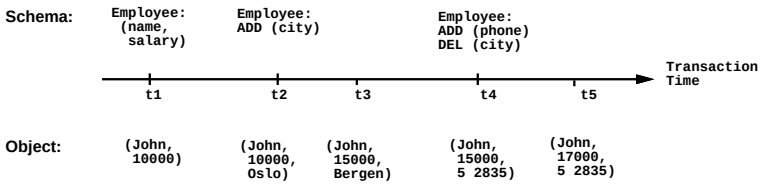

Figure 3.1: Schema Versioning – Transaction Time

This kind of schema versioning is in line with the definition of transaction time which is to record the change or transaction history of the database – one is not allowed to change the past because transaction time is reflecting the time when a database changes its state. For this kind of schema versioning see for example [20, 41, 55, 56].

On the other hand, by valid-time schema versioning both the current as well as previous schema versions may be altered and changed. A single change statement may alter one or multiple versions of the schema, for example, the addition of an attribute definition to an object type shown in Figure 3.2. We use the same example as of Figure 3.1, where the "Schema before" part in Figure 3.2 illustrates the corresponding valid times of the schema changes illustrated by Figure 3.1 where . Thus, there are three schema versions valid at distinct and non-overlapping intervals. However, in retrospect we "now know" that all employees actually had an employee number (emp_no) during a certain period of time, say between times k' and k''. We want to add this fact to the database by a change statement. The fact spans several of the existing versions of the schema. That is, several versions are altered by a single statement. The result of this change statement give rise to five versions versus three before the change statement was issued, see the "Schema after" part of Figure 3.2.

The benefit of a valid-time schema versioning is that both schema and data are subjects to change and that the database better captures the facts as they appear(ed) in reality. That is, intentional changes to a database may

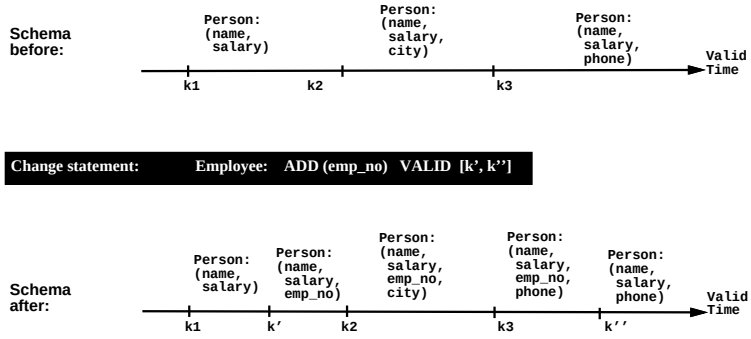

Figure 3.2: Schema Versioning – Valid Time

support both retro-active and pro-active updates by valid-time schema versioning. In this example we made a retro-active modification affecting several previous valid-time versions. By only transaction time support the knowledge of updates being either retro-active or pro-active is not maintained by the database. However, valid-time schema versioning for relational databases has yet got little attention by the research field. The only proposal known to us is presented by De Castro et al. [21].

We will not elaborate on the intentional issues any further. Our main emphasis will be on extensional issues of models and query languages.

3.4 User defined time

User defined time is regarded to be controlled by an application. In this section we present three ways of "incorporating" time without extending or changing the semantics of an underlying system with semantics typically introduced by a temporal system. The first only defines an additional set of basic time-related data types. The second extends an existing system with an additional "temporal" layer on top of it. The third is an example on how temporal data may be stored and accessed by a conventional system.

3.4.1 Adding Data Types

The first and simplest solution is to extend the set of basic data types with DATE, TIME, DATETIME and INTERVAL with some intrinsic operations, such as in SQL–92 [44] and the "pure" SQL3 standard[2] [45, 46]. However, the SQL/Temporal part of SQL3 includes proposals for both valid- and transaction-time support as extensions to the SQL3 standard [65, 64].

[2]The "pure" SQL3 standard defines a framework and a database language foundation as a self contained core of the standard. Other, non-mandatory, features are defined as separate parts of the standard.

Several commercial relational systems (RDBMSs) support one or more of the simple temporal data types. Although time is materialized as simple data types the systems do not support time in the sense what is meant by a Temporal DBMS. The handling of temporality and history of data, i.e., the time-varying and time-dependent aspects of data, is managed by the applications and the RDBMS has no built-in semantics for capturing these aspects of data.

Attempts have been made to incorporate some of the time-varying aspects of data into existing non-temporal RDBMSs without redefining the whole system and/or making versions of the systems too incompatible. Based on the tradeoffs between the responsibility of a DBMS and that of an application some functionality has been added to the DBMS or explored for the management and access of temporal data as presented in sections 3.4.2 and 3.4.3, respectively. Both rely on the existence of some time data type(s) of the underlying RDBMS.

3.4.2 Adding a 'Temporal' Layer

Vassilakis et al. present in [71] a transaction support[3] for a temporal extended RDBMS system. The temporal extension is realized as an additional layer, called the temporal engine, on top of the Ingres RDBMS. Thus, their main objective is to develop a transaction support for a temporal DBMS by utilizing the transaction features of the Ingres DBMS.

Furthermore, they extend the Ingres kernel with a data type DATEINTERVAL and operations on this data type, based on a pair of the DATE datatype. The temporal model is defined by means of tuple timestamping for valid-time, and its semantics is managed by the temporal engine. A timestamp could either by of data type DATE or DATEINTERVAL.

A valid-time relational algebra, termed VT-AL, is defined to support an efficient and temporal consistent data manipulations. The operators of this algebra is FOLD, UNFOLD, PUNION, PEXCEPT, and NORMALIZE. The first two of these are the most fundamental, and they are used to map relations between interval- and point-based timestamped representations (See also Lorentzos and Johnson's algebra [39] for similar operators). For example, when UNFOLD takes as input an interval timestamped relation, it returns a relation where each input tuple is replaced by a set of *value-equivalent* tuples[4] [31], i.e., every tuple in such a set has a point timestamp identical to a point within the interval timestamp of the original tuple, and any two

[3] Transactions are here regarded as transactions in a traditional database sense, e.g. according to the ACID properties, and should not be confused with a transaction-time mechanism in a temporal RDBMS.

[4] Value-equivalent tuples are tuples where each attribute value of a tuple is equal to the value of the same attribute of all the other tuples in the set (timestamp values need not to be value-equivalent).

value-equivalent tuples in the set has distinct point timestamps. FOLD is the "inverse" operator, for example, value-equivalent timestamped tuples, with identical or adjacent timestamps are collapsed into one tuple of the result relation, but FOLD ensures that every result interval timestamp covers all the time points defined by the timestamps of its operand relation.

The NORMALIZE operators are defined in terms of FOLD and UNFOLD, i.e., it is defined identical to first an UNFOLD operation followed by a FOLD operation. Both PUNION and PEXCEPT are temporal redefinitions of their SQL–89 counterparts. The PUNION is an ordinary UNION operator but the result is NORMALIZEd. The PEXCEPT is also an ordinary EXCEPT operator but the operands are first UNFOLDed, and the result is FOLDed.

At the user level SQL–89 is consistently extended both for data definitions and temporal queries, this extension is termed VT-SQL. Both queries and modification statements are extended with language constructs that capture the temporal aspects of data manipulated by the user. All temporal statements issued are pre-processed by the temporal engine before they are issued to the underlying Ingres DBMS.

The novel approach of this system is the integrated support of database transactions and temporality. The temporal transaction management layer (i.e., the temporal engine) uses a combination of allocating user sessions and system sessions to manage system internal temporary tables and objects locking, and also manages recovery in this way.

Other examples of the temporal layer approach are given by ATSQL of Böhlen and Jensen [9] that is a temporal extension to SQL–92 implemented on top of the Oracle DBMS, and the extended relational algebra of Lorentzos and Johnson [39] implemented on top of Ingres DBMS by means of the Quel macros.

3.4.3 Use of a Conventional System

Leung and Pirahesh show how to handle time-varying data for both storage and query evaluation by a DB2 RDBMS for AIX version 2 [38]. Especially the data type DATE and the recursive SQL of DB2 play central roles in representing temporal data and in defining two important operators respectively.

The approach is based on tuple timestamping either by supporting a time-point timestamp or a time-interval timestamp. The first is defined as a single DATE attribute of a relation, and the other as start-time and end-time DATE attributes, respectively. This is how temporality is defined and managed persistently. The DBMS does not know that these attributes denote timestamps.

Two important temporal operators are defined. One is the so-called *as_ of* or *time-slice* operator for access of data at a given time-point or time-interval. The other is a *time-coalesce* operator which is used to compress the number of tuples of a relation. These operators are defined and formulated by means of the existing DB2's recursive SQL – no language extensions are introduced to handle temporal queries.

The *time-slice* operator takes a relation and a time-value (i.e., time-point or time-interval) and returns a set of tuples with valid tuples for the specified time-value.

The *time-coalesce* operator takes all value-equivalent tuples of its operand relation and returns a relation where all value-equivalent tuples have non-overlapping and non-adjacent timestamps. That is, each output tuple is a result of collapsing a corresponding set of value-equivalent input tuples that have overlapping or adjacent timestamps. Hence, the timestamp of each result tuple is the union of the timestamps of its input tuples. Note the similarity between this operator and the FOLD and NORMALIZE operators presented above.

Storage-space considerations are also discussed. They propose two models, named incremental-forward and incremental-backward models. The former stores the complete initial state of an object, i.e., an initial tuple, and only changes to that object are added later on. That is, a change is a *new* tuple inserted but all other attributes beside the key attribute(s), timestamp(s), and the modified attribute(s), are given NULL values. The rational behind this approach is that NULL values normally occupy less space.

In the case of an incremental-backward model only the current state of an object is completely stored, and previous states are "filled" by NULL values in the same manner as in the forward model.

3.5 Tuple timestamping

The most common introduction of time into the relational model is by tuple timestamping, and where the query language and relational algebra is extended accordingly. All of the early temporal relational models concentrate on tuple timestamping [4, 6, 17, 63].

Over the past ten to fifteen years more than a dozen tuple timestamped temporal relational models and algebras have been proposed in the literature. We only present a few of these to indicate the differences in approach. In principle these approaches differ along the following lines. They may support valid-time, transaction-time, or both time-dimensions. Further, they support timestamps as either time-points or time-intervals. They may support only notions of past and current time, or they in addition support some notions of future time where timestamps are defined in terms of explicit values and/or variables like *uc*, ∞, and *forever*.

3.5.1 Basic Approaches

Section 2.3 presented the two time-dimensions of valid-time and transaction-time, respectively. To indicate different ways in managing temporal relations we show how a selected set of proposals define their respective temporal relational models. In the following we more formally present the two dimensions (or domains) for valid-time, D_v, and transaction-time, D_t. The time dimensions may be defined as follows (for simplicity we use discrete total ordered set of time instants to represent both time-dimensions):

- Transaction time-dimension and ordering:

$$D_t = \{t_0, t_1, \ldots, t_i, \ldots, now\}$$

$$\forall t', t'' \in D_t \setminus \{now\} \ : \ t' < t'' < now \ \vee \ t'' < t' < now \ \vee \ t' = t'' < now$$

- Valid time-dimension and ordering:

$$D_v = \{t_0, t_1, \ldots, t_i, \ldots, now, \ldots\} \cup \{\infty\}$$

$$\forall t', t'' \in D_v \setminus \{\infty\} \ : \ t' < t'' < \infty \ \vee \ t'' < t' < \infty \ \vee \ t' = t'' < \infty$$

For both time dimensions D_x and time values given in time instants and time intervals the following equations apply:

1. $t_k = t_0 + k \ and \ t_k = t_{k-1} + 1, \ k \geq 1$
2. $[t_k, t_l] = \{t_i | t_i \in D_x \ \wedge \ t_k \leq t_i \leq t_l \ \wedge \ t_0 \leq t_i\}$

The next definitions show how one could define different bi-temporal relational schemata, R_i, where the temporal information has domains defined by the time-dimensions D_t and D_v, respectively:

1. $R_1 = (A_1, ..., A_n, T_v, T_t)$
2. $R_2 = (A_1, ..., A_n, T_{v_s}, T_{v_e}, T_t)$
3. $R_3 = (A_1, ..., A_n, T_{v_s}, T_{v_e}, T_{t_s}, T_{t_e})$

where each A_i is a regular attribute of the relation (including key attributes), and T_{x_s} and T_{x_e} are start and end times of interval timestamps. Subscripts $_v$ and $_t$ stands for valid and transactions timestamps respectively.

Codd's definition is, informally, extended to $R = (A_1, ..., A_n | T)$, where the T is characterized by a combination of T_x's as shown above. Furthermore, each timestamp is either a name-point pair, (T_x, d_x), if a time-point timestamp is applied, or it is a name-interval pair, $((T_{x_s}, T_{x_e}), [d_{x_s}, d_{x_s}])$, when an interval is the timestamp.

We coin the three R_i's schemata for event-based, interval-event based, and interval-based bi-temporal models, respectively. Each of these is presented below with examples of models proposed in the literature.

3.5.2 Event-based models

The first, R_1, is an event based bi-temporal relation. That is, it does not consider for example the duration of when an object was valid in reality. It only states that at time $t_i \in D_v$, possibly some event occurred, the fact become valid, and was registered at time $t_j \in D_t$. Of course an application may impose additional interpretations if it knows that an event's duration is always from this time until the next event occurs for the same object, i.e., the next event that is "closest" in time. A problem arises with this inter-event relationship, because events may enforce different interpretations at different moments in time, say that a past event is always fixed in the past, but a predictive event may change because of its predictive nature. When predictive events become past events, an inconsistency may occur because the recorded sequence of events may be different to the actual sequence occurred in reality.

The lesson is that event relations should only act as pure event relations where the time to consider capture only instants of time and not some other derived time semantics, like durations or lifespans. The same applies to predictions of events in this context. If predictions are allowed, there should be some support for physical and/or logical deletion of predictions which change before they become present and past events. (For example, both Jensen [30] and Ben-Zvi [6] use specific attributes to record changes and errors respectively. See below.) By use of a bi-temporal relation a correct sequence of events may be detected by means of known relationships between valid-time and transaction-time. Sometime it is better to separate the past and present from the predictive data. In contrast to the other two relational schemata event timestamps do not use variables or distinguished individuals. Only explicit individuals are allowed as timestamp values.

3.5.2.1 Ariav's model

Ariav's model is a bi-temporal relational model where timestamps are based on time points [3].

name	*department*	*position*	*Valid-Time*	*Trans-Time*
Paul	design	assistant	Jan90	Feb90
Paul	design	assistant	Mar91	Dec90
Paul	design	engineer	Aug91	Dec91
Mary	design	manager	Jan90	Jan90
Mary	HQ	AD	Dec90	Feb91

Table 3.1: A bi-temporal Employee Relation by Ariav's model

The employee relation with six tuples of the relation of Table 2.2 in Section 2.3.3, will have a relation with only five tuples by Ariav's model, as

shown in Table 3.1. It is obvious that this representation misses the explicit information such that Mary was both a manager and an AD between December 1990 and February 1991 (see Table 2.2). The same is true for the fact that the database stored inaccurate information about Paul being an assistant between March 1991 and December 1991. Paul was only an assistant from March until May that year. That is, if someone wants to verify if the data used at a previous point in time was inaccurate or inconsistent with what we now at the present time, then this information is hard, if not impossible, to derive from a model of this kind. In both cases the information may be derived by an application, but that depends on that only the application knows how this information is to be derived from the data, and including that it provides a record for resolving all previous inaccuracies and inconsistencies.

3.5.3 Interval-event based models

The second, R_2, shows a mixed interval-event based bi-temporal relational schema. It supports actual semantics of the duration of some fact in reality. In addition the semantics may say something about the lifespan of an object. That is, when the object was defined, when it eventually disappeared, and when it was redefined. In addition to past and present these relations may also store predictive data. The transaction-time support is given by time points. All data is registered with a when stored timestamp. Hence, the database captures no semantics of the period of time when the fact was current in the database, only when it became current.

3.5.3.1 Jensen's Model

name	*department*	*position*	*Valid-Time*		*Trans-Time*	*Op*
Paul	design	assistant	Jan90	Sep90	Feb90	*I*
Paul	design	assistant	Jan90	Sep90	Dec90	*D*
Paul	design	assistant	Mar91	Nov91	Dec90	*I*
Paul	design	assistant	Mar91	Nov91	Dec91	*D*
Paul	design	assistant	Mar91	May91	Dec91	*I*
Paul	design	assistant	Mar91	May91	Dec91	*D*
Paul	design	engineer	Aug91	*uc*	Dec91	*I*
Mary	design	manager	Jan90	Feb91	Jan90	*I*
Mary	design	manager	Jan90	Feb91	Feb91	*D*
Mary	HQ	AD	Dec90	Dec93	Feb91	*I*

Table 3.2: A bi-temporal Employee Relation by Jensen's model

Jensen's model [30] is an example of an interval-event model. Table 3.2 takes the relation of Table 2.2 in Section 2.3.3, and represents it by Jensen's

model[5]. The *Op* attribute denotes tuples as deletion and insertion requests indicated by *'D'* and *'I'*, respectively. Modifications of an object is handled by a pair of deletion and insertion requests in the following manner; First a deletion request is represented by a new tuple with the same attribute values and valid-time as its previous insertion request of this object, but the *Op* value equals *'D'*; Second, the deletion request is followed by an insertion request with a transaction-time identical that of the deletion request it is paired with, but in this case the *Op* value is equal to *'I'*.

It is not clear if two tuples, such as in Mary's case, are allowed to record information about the same object with overlapping valid-time intervals. Because an I-marked tuple with no corresponding and succeeding D-marked tuple is regarded as the only present/latest valid state of the object. So, information where two facts about an object are true at the same time has to be recorded after one another. That is, the "latest" fact of an object must always be represented by a retroactive deletion-insertion pair of some previous fact. A similar problem arise with Paul's inaccurate information of he being an assistant between March 1991 until November the same year. This inaccuracy is not explicit in this type of representation. That is, data that records an error and data that only supersedes old data are treated the same way, and the application has to know which one is what, if that is possible at all.

3.5.4 Interval-based models

The last schema, R_3, shows an interval based bi-temporal relation. Both intervals denote a start time and an end time. The transaction-time registered the duration when a tuple is current in the database. There are different interpretations of a transaction interval timestamp due to handling of variables. The information of a timestamp pair, $([v_s, v_e], [t_s, t_e])$, of valid- and transaction-times respectively, is defined as follows:

1. If t_e is a variable, e.g. *uc*, then the tuple is current in the database
2. If $t_e = t \in D_t$ then the tuple is logically deleted
3. If $t_s < v_s$ then the tuple is pro-actively inserted
4. If $t_s > v_s$ then the tuple is retro-actively inserted
5. If $t_s = v_s$ then we have a "synchronized" insertion

[5] Jensen's model is a so-called backlog representation. That is, the actual representation of the data is divided between a table handling current data and a backlog table handling previous data.

3.5.4.1 Snodgrass' model

Snodgrass' TQuel (a temporal extension of the Ingres Quel) is an example of a query language which is based on a (relational) model with interval timestamping both for the valid-time and the transaction-time [63]. A relation illustrating Snodgrass' model is given by the Employee relation of Table 2.2 in Section 2.3.3.

3.5.4.2 Ben-Zvi's model

Another approach for defining interval tuple timestamping is given by Ben-Zvi's model [6, 25]. Ben-Zvi was the first to introduce different times; called effective time and registration time, which are analogous to our notion of valid time and transaction time respectively. A relation schema, R, is on the following form:

$$R = (A_1, ..., A_n, T_{es}, T_{rs}, T_{ee}, T_{re}, T_d)$$

where the A_i's are as in the previous cases, and:

- T_{es} and T_{rs} are the first pair of timestamps and define effective start-time and registration start-time, respectively, which correspond to T_{v_s} and T_{t_s} in the previous examples.
- T_{ee} and T_{re}, the other pair, define the effective end-time and registration end-time, respectively. The T_{ee} corresponds to T_{v_e}. The time T_{re} denotes the time when the time T_{ee} is recorded. Both T_{ee} and T_{re} may be given as '-', a symbolic value denoting, at present, an unknown time.
- T_d is used to differentiate between changes and errors. If the T_d attribute has a timestamp, the information captured by the tuple is known to be an error and therefore logically deleted. Otherwise, the T_d's value is '-', and then the tuple is regarded as a current fact, i.e., a latest or previous fact about some object stored in the database.

If we employ the employee relation of Table 2.2 in Section 2.3.3, which conforms to Snodgrass' temporal model [63], we have the relation shown in Table 3.3.

Beware that Ben-Zvi's model gives another definition of the concept of a transaction time. In this model registration times, T_{rs} and T_{re}, record when facts have known effective start- and end-times, respectively. In contrast, transaction (interval) timestamps denotes when facts are current (i.e., accessible) in the database. Ben-Zvi's definition relies exclusively on the time T_{re}, that is defined relatively to the time T_{ee}. This is a significant difference from the definition of what is regarded current (i.e., accessible) in respect to the notion of a transaction time (interval) [31]. For example, in Snodgrass'

name	*department*	*position*	T_{rs}	T_{es}	T_{ee}	T_{re}	T_d
Paul	design	assistant	Jan90	Feb90	Sep90	Feb90	'-'
Paul	design	assistant	Mar91	Dec90	Nov91	Dec90	Dec91
Paul	design	assistant	Mar91	Dec91	May91	Dec91	'-'
Paul	design	engineer	Aug91	Dec91	'-'	'-'	'-'
Mary	design	manager	Jan90	Jan90	Feb91	Jan90	'-'
Mary	HQ	AD	Dec90	Feb91	Dec93	Feb91	'-'

Table 3.3: A bi-temporal Employee Relation by Ben-Zvi's model

model [63] an explicit transaction end-time denotes when a fact was recorded as logically deleted. In Ben-Zvi's model this is solely denoted by the means of the T_d attribute.

3.5.5 First Normal Form and Object Identity

All tuple timestamped relations obey the first normal from (1NF) restriction, i.e., no multi-value attributes are allowed. But what is regarded as an entity (or object) in reality may be defined by several tuples in a database, i.e., database objects. Hence, it is not necessary a one-to-one correspondence between an entity in reality and an object in the database. The entity is represented by several tuples. This set may have redundant information because some attribute values may for example change more frequently than others. This, in turn, causes the less frequently changed attributes to replicate, at least logically, their values over several tuples.

The 1NF relational databases have no intrinsic mechanism to automatically capture changes to an object's identity, i.e., the primary key of a tuple. When a primary key of a tuple is changed the temporal database inserts a new tuple with the new primary key, but the database regards the new inserted tuple as representing a different object. That is, the database cannot deduce that these two tuples represent the same object. This lack of "dynamics" is inherited from the 1NF snapshot relational model, hence, applications have to manage changes of this kind. In Section 3.6 we will see that attribute timestamped relational models to some extent handle these shortcomings of the relational model.

3.6 Attribute timestamping

Although most temporal relational models are tuple timestamped there are some proposals for attribute timestamping. In this approach a time-variant attribute is not a single value but a multi-value, or more correct a complex value like a function with domain time (or a Cartesian product of times), where its range is the value domain of the attribute timestamped. Hence,

this approach breaks the first normal form restriction on relations. Relations of this type are known as Non-1NF (N1NF) relations.

3.6.1 Basic Approaches

Three different approaches on attribute timestamping could be defined as follows:

1. $R_1 = (T_t, R'),\ where\ R' = (A_1 T_{v_1}, ..., A_n T_{v_n})$
2. $R_2 = (\{(A_1, V \times T)\}, ..., \{(A_n, V \times T)\}),\ where\ V = [T_{v_s}, T_{v_e}]\ and\ T = [T_{t_s}, T_{t_e}]$
3. $R_3 = (\{(A_1, T_v)\}, ..., \{(A_n, T_v)\}$

where A_i's are attributes of the relation (including key attributes), and T_{x_s} is start time and T_{x_e} is end time, both associated with interval timestamps. Subscripts $_v$ and $_t$ stand for valid and transactions timestamps, respectively. These three approaches are named transaction-time indexed models, bi-temporal multi-value models, and valid-time multi-value models, respectively. Representatives of each of these models are presented in the following sections.

3.6.2 Transaction-time Indexed Models

The first schema, R_1, defines a relation consisting of transaction-time T_t and a valid-time relation R'. That is, each transaction-time timestamp is an index for a separate valid-time relation (instance) of the relation scheme R'. We say that changes made to a relation of type R_1 are indexed by transaction-times, and, hence, each such index captures a transaction-time state of the relation R_1. Thus, the index could be thought of as a stack of transaction-time slices.

3.6.2.1 McKenzie's Model

McKenzie's model [40, 43] is a transaction indexed approach. A temporal database, $\mathbf{D_{db}}$, in McKenzie's model is a finite sequence, $D_{db_0}, ..., D_{db_n}$, where each element, D_{db_i}, of the sequence is a database "state" relative to a particular transaction time.

An example is the Object Salary relation (Table 2.1) presented in Section 2.3.3. This relation translated to the representation of McKenzie's model is shown in Table 3.4, where it is represented as a sequence of valid-time relations each of which is indexed by a unique transaction timestamp.

The Object Salary relation, Table 3.4, is represented by tuples constituted by single transaction-time instants and a relation of valid-time attributes. The valid-time of each individual attribute is given by a set of time

T_t	R'
0	$\emptyset$
Apr88	{(15000, {Jan88,...,Dec89})}
Mar90	{(15000, {Jan88,...,Dec89}),(17000, {Jan90,...,Mar92})}
Oct94	{(15000, {Jan88,...,Dec89}), (17000, {Jan90,...,Mar92}), (19000, {Sep94,...,Dec97}), (25000, {Jan98,...,∞})}

Table 3.4: Object Salary by McKenzie's model

points. The times is of type *mmyy* and a chronon is one month. McKenzie's model does not handle modifications of facts satisfactory. For example, when a fact is to be deleted from a valid-time relation, $r_i(R')$, it is simply done by logically inserting a new valid-time relation, $r_{i+1}(R')$, with the fact removed, indexed by the new transaction-time t_{i+1}. The information that a fact was deleted cannot be deduced without comparing one valid-time relation with its previous and/or following valid-time relations in the sequence. The second drawback is that changing a key attribute value of a fact will have the same effect as with 1NF relations. The third drawback is that the expected spinoff of reduced redundancy is not obviously achieved, because each attribute value of a valid-time relation is single-valued, i.e., they are replicated over several tuples indexed by different transaction times.

3.6.3 Bi-Temporal Multi-Value Models

The second schema, R_2, has multi-valued attributes where each multi-value is a triplet of a valid-time (e.g. interval), a transaction-time (e.g. interval), and a value of the attribute domain. A tuple of n attributes is given by n sets of such triplets, each set representing the bi-temporal information of an attribute of that tuple at different times.

3.6.3.1 Gadia and Yeung's Model

Gadia and Yeung's model [26], called the heterogeneous model, is a representative of R_2.

The schema of the relation in Table 3.5 is a specialization of the schema given for the relation of Table 2.1 (the tuples are not related). We have included an employee attribute to make this illustration more interesting, i.e., Paul and Mary are included as employees and the relation records data about salary histories of employees. Beware that the original salary relation shown in Table 2.1 would have had only one attribute (one column in the table) by Gadia and Yeung's model. This model copes with the redundancy problem, and it also allows existing keys to change values. In this example Paul's *Name*, which is the key attribute, is changed from Paul to Peter in September 1991, but valid from August 1991.

Name	*Salary*
Paul, [Jan90,Sep90]×[Feb90,*uc*]	30000, [Jan90,Sep90]×[Feb90,*uc*]
Paul, [Mar91,Nov91]×[Dec90,Dec91]	30000, [Mar91,Nov91]×[Dec90,Dec91]
Paul, [Mar91,Jul91]×[Dec91,*uc*]	30000, [Mar91,May91]×[Dec91,*uc*]
Peter,[Aug91,*uc*]×[Sep91,*uc*]	35000, [Aug91,Dec92]×[Dec91,*uc*]
	40000, [Jan93,*uc*]×[Dec91,*uc*]
Mary, [Jan90,*Dec*93]×[Jan90,*uc*]	60000, [Jan90,Feb91]×[Jan90,*uc*]
	100000,[Dec90,Dec93]×[Feb91,*uc*]

Table 3.5: Object Salary by Gadia and Yeung's model

3.6.4 Valid-Time Multi-Value Models

A valid-time multi-value model, R_3, is a simplification of the previous model, R_2, in the sense that it only supports valid times and not transactions times. Some representatives of this approach are briefly presented, because each one of them provided novel research contributions.

3.6.4.1 Gadia's Model

Gadia's model [24] is an example of this approach defined over the valid-time dimension. The intrinsic time of Gadia's model is based on interval timestamping.

The novel contribution of this model is the homogeneity assumption (Section 3.7) imposed on a database. A homogeneous temporal tuple has all its attributes defined over the same time period(s). In other words all attributes of a tuple have identical timestamp values. Gadia's idea was to prevent objects containing NULL-values, i.e., all attributes have to be defined during the lifetime of an object.

3.6.4.2 Tansel's Model

Tansel's model [67] adds both valid-time and different structure types on attributes, such as atomic, set-valued, triplets, and set-triplet-valued attributes. The first contains atomic values only. The second is a set of atomic values. The last two are attributes with timestamps. That is, each triplet is an attribute value with valid start- and end-times.

The model suggests to use both atomic and triplet attributes for keys where values are not expected to change over time. If they do change, a new tuple has to be inserted and the redundancy problem is in some sense analogous to that of tuple timestamping. Tansel's contribution is the combinations and co-existence of different attribute structures in the same relation.

3.6.4.3 Clifford and Croker's Model

Clifford and Croker's model [12] is based on representing attributes as pairs. That is, a function takes a time point of the time domain and returns an element of the value domain of the attribute, i.e., the attribute value defined at that time.

The novel contributions of this model are the notions of *function values*, as mentioned above, and the algebra based on *lifespans*. (The notion of a lifespan were first introduced by Clifford and Tansel [15].) Lifespans are used to capture the existence of database objects, i.e., including "birth", "death", and "rebirth" of such objects. Both tuple and attribute lifespans are defined. The former captures lifespans on the data instance level, i.e., of tuples. On the other hand, an attribute lifespan reflects when a particular attribute value was/is defined. Since lifespans are sets of time points set-theoretic operators apply to these structures.

Name	*Salary*	*Lifespan*
Jan90 $\mapsto$ Paul	Jan90 $\mapsto$ 30000	
...	...	
Sep90 $\mapsto$ Paul	Sep90 $\mapsto$ 30000	
Mar91 $\mapsto$ Paul	Mar91 $\mapsto$ 30000	
...	...	
Aug91 $\mapsto$ Peter	Aug91 $\mapsto$ 35000	
...	...	$\{Jan90, ..., Sep90\} \cup$
Jan93 $\mapsto$ Peter	Jan93 $\mapsto$ 40000	$\{Mar91, ..., now\}$
Jan90 $\mapsto$ Mary	Jan90 $\mapsto$ 60000	
...	...	
Dec90 $\mapsto$ Mary	Dec90 $\mapsto$ 10000	
...	...	
Dec93 $\mapsto$ Mary	Dec93 $\mapsto$ 10000	$\{Jan90, ..., Dec93\}$

Table 3.6: Object Salary by Clifford and Croker's model

Table 3.6 illustrates how the Object Salary relation of Table 3.5 is translated to Clifford and Croker's model. Each multi-valued tuple has a *lifespan*. Even the first tuple has a lifespan of two not adjacent time periods. The functional aspects of this model is defined such that at each time point within a tuple's lifespan the function returns the attribute value of the tuple at that time.

Gadia and Yeung's [26] models is functional too in the same sense as Clifford and Croker's model.

3.6.5 Non-First Normal Form and Object Identity

Two obvious advantages may be gained by attribute timestamping. Firstly, by means of complex or multi-valued attributes an object in reality may be

represented by only one tuple in the database, and one-to-one correspondences between entities in reality and database objects are established at the database level. This property increases the modeling power. A side effect of attribute timestamping is that primary keys are in some models allowed to be time-variant. Many applications may require that a key does not need to be constant over time. Still, all attributes, keys included, should be unambiguous at any given time instant.

Secondly, when properties of an object change at different times and/or with different frequency, only the individual attributes involved are affected, and not the tuple as a whole. Only the new attribute value(s) and timestamp(s) are inserted. The problem of a potential redundancy as introduced by tuple timestamping may be avoided by this approach. But, for example Mckenzie's model presented above does not reflect the redundancy issue, i.e., each attribute value is logically replicated over all transactions-time indexed states as long as it is valid.

3.7 Query Language

By incorporating the temporal dimensions into the relational model the semantics of time become part of the underlying temporal relational model. To take advantage of the inherent semantics of temporality in querying a database, the query language, the relational algebra and the calculus have to extend constructs, set-theoretic-like operations, and semantics to capture the inherited nature of temporal data.

3.7.1 Introduction

In the case of user-defined times no other extensions are required than to make intrinsic operations like equality, less-than, etc. applicable on basic data types for time (e.g. DATE, TIME, and DATETIME). This is done in the same manner as with the operations applied on basic data types like INTEGER, STRING, etc. In the cases of tuple and attribute timestamping more comprehensive considerations have to be taken into account. For example, the snapshot employee relation of Table 3.7 records name, department and positions of each employee.

name	*department*	*position*
Paul	design	assistant
Paul	design	engineer
Mary	design	manager
Mary	HQ	AD

Table 3.7: A Snapshot Employee Relation

Querying this snapshot relation: *'determine the positions held by employees of the design department'*, or expressed in snapshot SQL:

```
SELECT  position
FROM    employee
WHERE   department = 'design'
```

The result relation consists of the following three 1-tuples: (assistant), (engineer), (manager). However, imagine that the same query is issued on the relation of Table 2.2 in Section 2.3.3, then the result is a valid-time relation shown in Table 3.8.

position	*valid-time*	
assistant	Jan90	Sep90
assistant	Mar91	May91
engineer	Aug91	*uc*
manager	Jan90	Feb91

Table 3.8: A valid-time result relation

Obvious, a syntactical identical query of the temporal SQL could be defined by a temporal semantics, i.e., as the default semantics, as in the above example. If we only wants information of the current or some other snapshot state, we have to state that explicitly, and, then, we deal with what is termed a *reference time*—to be discussed next.

3.7.2 Time Reference in Queries

We can now extend the complexity of querying the database by introducing what may be called a time scope, a time window, or a time reference. If, for example, we want to know the state of the database at a particular time in the past, we may query the relation of Table 2.2 in Section 2.3.3: *Find all positions held by employees of the design department as of July 1990.* The result valid-time relation is shown in Table 3.9.

position	*valid-time*	
assistant	Jan90	Sep90
manager	Jan90	Feb91

Table 3.9: A valid-time result relation at a particular reference time

By introducing the notion of a reference time, we can both view the reality and the database as they were at some other states states. Beware that we can define a reference time relative to either valid-time or transaction time, i.e., *'as_of'* and *'as_best_known_as_of'*, respectively. The above example was a *'as_of'* query with a transaction reference time automatically

set *'as_best_known_as_of'* the time *now*. If no reference time(s) is (are) specified the default for valid-time is all times, i.e., the whole history, and the corresponding transaction-time default is *now*.

For example, *'as_best_known_as_of'* July 1991 the database stored information that we now know was inaccurate. The information that an assistant position is held until November the same year is wrong, i.e., Paul holds his assistant position only until May that year. By means of a reference time we may actually investigate or analyze the database at previous times, and then refine earlier reports and decisions based on this information. The term *reference time* was introduced by Clifford and Isakowitz [18], but its notion is based on an earlier work by Finger [23] who takes an analogous approach on historical databases as that of "point of reference" in intentional logic by Montague [47].

3.7.3 Definitions and other Properties

Before going into detail on the various models and approaches regarding their extensions on their operational parts we consider a few important definitions. A *homogeneous tuple* is a tuple where each attribute has the same *temporal domain* [24], i.e., each timestamp of each attribute is the same. The temporal domain of an attribute is a temporal element which is a finite union of intervals [24]. (Beware that the terms *temporal domain*, *temporal element*, and *lifespan* denote the same concept. We use hereafter only the term *lifespan* as introduced by Clifford and Tansel [15].)

The *homogeneity assumption* is introduced by Gadia [24] to be the temporal counterpart of a conventional relational database without NULLs. That is, when the attribute lifespans of a tuple are not identical, then a snapshot of this database at time, say t_i, may result in producing NULLs for some of the attributes. This is due to the obvious reason that some attribute values are unknown at time t_i. This is the case in the heterogeneous model by Gadia and Yeung [26]. The homogeneity assumption may be enforced both on time dimensions, relations, or the whole database. In the tuple timestamped approach a tuple is by definition homogeneous when NULLs are not allowed.

The relational algebra, as defined by Codd [19], operates on only one object type, namely the relation. That is, both the domain and range of an algebraic operator are relations, though, of possibly different kinds. The five fundamental operators of this algebra are: union, set difference, Cartesian product, projection, and selection, and, for example, operators such as intersection, join, division are all defined in terms of these five fundamental operators.

A temporal language needs also to specify time both as domain and/or range for its set of operators. Time-values like points and intervals are elements and subsets of the domain D_x, respectively. That means that time entities like intervals and lifespans, if they are interpreted as sets of time

instants, are closed under the set-theoretic operations of union ($\cup$), intersection ($\cap$), difference ($\setminus$), and complementation ($\neg$). The semantics of these operators are then given by their set-theoretic semantics.

3.7.4 Temporal Algebras - Their Objects and Operators

Most temporal models and algebras differ in the type of objects they define and on the operators they provide on objects, respectively. See McKenzie and Snodgrass [42] for a presentation and a more comprehensive evaluation of twelve algebras incorporating time. The following is in part a brief summary of that paper.

3.7.4.1 Objects

Three kinds of objects are proposed: snapshot relations, temporal relations, and lifespans. Five of the algebras support snapshot relations [6, 39, 40, 48, 59].

Only one of the algebras, namely that of Lorentzos and Johnson [39], does not support time-oriented objects. This model and algebra define timestamps explicitly as user-defined times, but introduce new operators, UNFOLD and FOLD, to map between an interval representation and point representation of timestamps, respectively. Each interval timestamped tuple is expanded by UNFOLD into its point-based counterpart. That is, an interval-based tuple is expanded into several tuples, where all the new tuples hold the same information (i.e., they are value-equivalent), but each defined at unique and distinct point of the input interval. A temporal query imposes the following operational scheme: First, one UNFOLD the interval user-defined time relation(s) and then perform the wanted operation(s), and, then, the result is FOLDed back to an interval representation. In this manner the algebra uses the basic algebraic operators on the point-based relation to express the temporal queries.

Eleven of the algebras do indeed support temporal relations. Of these eleven three also manipulate pure time objects such as lifespans [12, 24, 26], e.g. these models have attributes defined as functions that map timestamps onto the attribute domain.

3.7.4.2 Operators

The set of algebraic operators defined by the non-temporal relational algebra is tailored to capture temporal operations. There are at least four classes or approaches of defining operators that could be distinguished. But, note that algebra proposals found in the literature may combine several of these classes. Thus, the distinction we make is solely to classify operators and not whole algebras.

The first class where the five basic algebraic operators are retained. For example, in the case of both Lorentzos and Johnson's algebra [39] and Vassilakis et al. VT-AL algebra [71], new operators are defined to capture the wanted temporal queries implicitly, like the UNFOLD and FOLD operators. There are to varying degrees how these operators have formal defined semantics. Some are only informally defined by examples.

The second class introduces what is called *snapshot reducibility* semantics [63]. The algebra defined for the homogeneous model by Gadia [24] and the algebra defined for ATSQL by Böhlen and Jensen [9] both satisfy this criterion. That is, applying a temporal algebraic operator on relation(s) at time t_i should produce the same result as applying its conventional algebraic counterpart to the snapshot version at time t_i.

The third class extends the set of conventional algebraic operators to capture time-varying information. The common way of doing this is to manipulate the input timestamps in such a way that the operator produces a meaningful timestamped output relation. For example, in Clifford and Croker's algebra [12] the lifespan of an output tuple produced by a Cartesian product is the union of the lifespans of the input tuples. Moreover, if an attribute of an input tuple is not defined for a time point of the output tuple's lifespan, then this attribute is assigned a null value at that time.

The last class of operators is the class of new operators. These operators could be used alone, others are defined to support the temporal extensions of queries in general.

Below we present some of the operators that we believe will illustrate the flavor of having temporal operators. Those presented here are defined by Clifford and Croker [12], but similar and analogous operators exist. Recall that Clifford and Croker's model is attribute timestamped, and both attributes and tuples have lifespans.

Time-slice The first is the unary *time-slice* operator, which is an operator that projects the temporal domain of the operand. Two time-slice operations are possible, one static and one dynamic. The static approach specifies a fixed lifespan condition, and returns all tuples that are defined for all points within the fixed lifespan. The dynamic approach specifies an attribute dependent lifespan conditions, and returns all tuples where each tuple is defined by the lifespan of its corresponding attribute's lifespan. The former returns tuples with identical lifespans, the latter returns tuples with, possibly, different, lifespans. This operator reduces the relation in the temporal dimension.

Select-when The next is the unary *select-when* operator, which takes a relation as input and produces a relation where each tuple's lifespan is restricted by the non-temporal selection condition, i.e., the result lifespan of a tuple is denoting exactly those time points when the selection condition is

satisfied. This operator reduces the relation in both the value and temporal dimensions.

Select-if A *select-if* is also a unary operator. It selects each tuples from a relation that satisfies a specified selection condition within a given lifespan. A select-if operation may be specified so that the result tuples satisfy the selection condition either at some point or at all points, i.e., existential and universal quantification, respectively. This operator reduces a relation only in the value dimension, and the lifespans of the input tuples are left untouched.

Time-join The *time-join* is a binary operator and uses as the join-condition a timestamped attribute of the first operand relation. The lifespan of each of the output tuples is the intersection of the lifespan of the condition attribute and the lifespans of the joined input tuples. If a resulting lifespan is empty (a lifespan is a set of time points) the tuple is not included in the result relation. This join operator imposes no relationship restriction on any of the value domains (i.e., the regular attribute domains) of the operand relations as would have been the case for the temporal extended θ-join, equi-join and natural-join operators. Hence, in the time-join case only the time dimension is restricted.

3.7.4.3 Conventional vs Temporal Queries

There are several reasons for defining a temporal algebra. The most obvious reason is that the conventional algebra as defined by Codd is unable to handle time-varying information as such. For example, assume that the relation in Table 2.2 has the valid-times as explicit user-defined attributes, and denoted by *From*-time and *To*-time attributes. (The transaction time is of no interest here.) If the query *'when was Mary an employee'* is issued, then the result should be Mary's employee history. The query is a projection on the *From* and *To* attributes.

From	*To*
Jan90	Feb91
Dec90	Dec93

Table 3.10: Mary's employee history

The result relation of the above query, see Table 3.10, shows two tuples overlapping on their valid-times, but we have no indications why it is like this. Are there two employees with name Mary? Does the input relation contain inconsistent data about Mary?

A temporal model and algebra would have recognized these overlapping intervals (or adjacent intervals) and coalesced the two tuples into one, and, hence, produce the correct result relation of one tuple with the interval [Jan90,Dec93], that in fact is the period when Mary was an employee in our example.

However, not all temporal query languages would automatically yield a coalesced result, because when objects are interval timestamped there could be reasons for retaining the interval structure for the result. For example, the result of the query: *when was Gro Harlem Brundtland prime minister of Norway*, is denoted by three distinct and non-overlapping periods, namely 1981, 1986–1989 and 1990–1996, regardless of coalescing or not. However, the result of the query: *find all periods when Bill Clinton was the President of USA*, would yield different results with and without coalescing. When the system imposes an automatic coalescing the result denotes that Clinton was president only during one period, namely 1992–2000. The correct result should be the periods 1993–1996 and 1997–2000. Of course, the above example and results could be interpreted correctly by users, because we may know that a single president period is only four years. This is a naive example, but there should not be difficult to envision situations that are more complicated and involved, and where a user could not deduce that some intervals have been inconsistently coalesced. The following, still naive, example illustrate this.

name	*department*	*salary*	*valid-time*	
John	Design	15000	Jan90	Sep91
John	HQ	15000	May90	May91

Table 3.11: A Salary History Example

We issue the following query on the relation of Table 3.11: *determine John's salary history*, stated as a temporal SQL-like query:

```
SELECT  name, salary
FROM    employee
WHERE   name = 'John'
```

This query would yield a relation with one tuple, namely (John, 15000, Jan90–May91). But, say that the meaning of John's tuples is that John had two jobs during the period of May91–Sep91, and, therefore, the correct salary for this period should be 30000, and not as 15000 as given above. In this example the result is an incorrect result due to automatically coalescing of result tuples.

ATSQL provides a explicit user-level language construct to enforce coalescing on the result [9], and thereby let the user decide if it make sense to apply this operation or not.

In general by defining formal temporal algebras query languages like SQL could be extended and assigned the formal semantics for operating on temporal data.

3.7.4.4 Optimization and Efficiency

A temporal algebra is needed for the system internal organization and behavior of a DBMS. No commercial relational DBMS provides an algebra as the query language. The most common query language is SQL (and Quel for the Ingres DBMS).

Unlike SQL, that is a declarative language, algebras are procedural. That is, the operational aspects are important and are used to optimize database queries that are either predefined or interactive. The conventional way of optimizing queries is to translate declarative statements (i.e., queries) into their algebraic counterparts. When a query is on an algebraic form the sequence of subqueries (i.e., partial queries), which constitute the original query, may be reorganized to reduce the number of internal operations (e.g. number of I/O accesses) or otherwise minimize the computation. Thereby, an optimization also decreases the user's response-time. The same should apply to temporal databases, and several studies and proposals on issues related to algorithms for temporal query optimizers have been published [1, 5, 10, 27, 28, 29, 37].

There is also another important and related aspect of having a temporal algebra which operates on the temporal structure imposed by a temporal model. That is, the system internal structures such as index structures. If an algebra does not handle temporality an optimization algorithm may not produce efficient expressions that could take into account system supported temporal indexes, storage structures, and so forth, for example see [2, 5, 22, 29, 36, 58].

3.7.5 Uniform Data Access

The reason for having a temporal model was to incorporate temporal data and their semantics, that are otherwise managed by each of the applications using the database, and providing a uniform model and management of the temporal data.

The reason for having a temporal query language is similar; accessing a database utilizing the potential advantages of temporal data makes the resulting objects (e.g. temporal relations, lifespans, snapshot relations) not only consistent with the modeled reality, but equally important queries and results are uniformly managed across applications. That is, both the temporal access strategies and temporal data independence are uniformly maintained by one (database) system for all applications. In consequence, temporal queries with identical semantics issued to the same database by different

and independent applications would yield the same result. In contrast, if the database had no built-in temporal support the temporal queries issued by different applications have to be identical syntacticly to ensure the same result, i.e., semantics implicitly defined by a query is unknown to the database.

We may regard each application as representing some concrete database view of the more "abstract" temporal data model defined by the actual database. What we then identify as *uniform data access* is in some sense analogous to Chomicki's notion of *representation-independence* [11].

3.7.6 Point- or Interval-based Query Languages

A query language can be point based or interval based. That is, to denote time by timestamps we either use points, i.e., time instants or intervals. Many query languages, including TQuel and TSQL2, adopt interval-based timestamps.

The most prominent reason for adopting interval as the "unit" of time is to efficiently encode sets of time instants, where each such set is associated with some data value. Thus, an interval is a practical representational solution to deal with, among other issues, space-efficient storage.

On the other hand a point based data model would (logically) replicate each data value at all time instants for which it is defined to be valid. This approach is not space-efficient and is regarded as less practical. Event-based models, such as presented in Section 3.5.2, are defined so that each data value is associated with only one time instant, and, therefore, event-based models cannot generally replace a point-based approach. Put differently, a point-based timestamp is a set of points and not a single point. An obvious drawback with a pure point-based model and language is that value equivalent tuples are always coalesced as a result of a query, i.e., one timestamp may denote different distinct periods of an object, but the initial structure of these periods are lost. This is a similar problem as discussed above with coalescing interval timestamps in query results.

There also are some servile problems with interval-based query languages and data model supporting interval timestamps. As mentioned an interval is nothing but some encoding of a set of time instants. Thus, when the query language also uses intervals, it is more reflecting the representational issues of the data rather than the semantics of the data. Formulating queries in an interval-based language have to take into account the encoding of the set of time instants they actually denote. And; whereas we are able to easily express queries in a point-based logic, their interval-based counterparts are not necessarily easily obtained, see examples in [69]. The effect is that declarative languages based on intervals have to deal, in some way, with the encoding of intervals. Hence, the pure declarative property of such languages are to some extent lost.

The semantics of a language are better handled by a point-based ap-

proach, because the semantics are better suited to be defined by some first-order (temporal) logic. The reason for this is that variables in a temporal logic refer to time instants as known individuals. In a corresponding logic for an interval language an explicit and special treatment of the notions of interval's upper and lower bounds are required, i.e., the logic has to manipulate the structure of the values, making expressions in such a logic rather cumbersome.

However, there are relational languages that support a mix of a point- and interval-based approach in both processing and manipulation of temporal data, e.g. ATSQL [9] and STSQL [8] but their query results are always interval-based. In particular, ATSQL applications can decide to coalesce or not to coalesce results of queries, and thereby retain the interval structure of the domain relation(s).

Chapter 4

Summary

We have presented a survey of temporal relational models and given an introduction to temporal query languages and their properties. The temporal relational models deal both with time-varying schema and data, coined in this report temporal intentional and temporal extensional, respectively. Both schema and data may be temporal relative to more than one time dimension. Time dimensions are orthogonal and impose different semantics on the timestamped objects.

Temporal extensional models are grouped according to three main approaches incorporating time as an integral part of the database. One is simply to introduce additional basic time types, and is classified as user-defined time. The other two are more fundamental with respect to what characterize temporal databases, and they mainly differ on which objects are timestamped. One timestamps tuples, the other attributes. However, models within each of these groups also differ in how they define and associate timestamps to their respective objects.

The temporal extensional models support what is commonly named history/predictions and revision of data. The former reflects the evolution of an object along the valid time dimension, the latter reflects the evolution of an object along the transaction time dimension, i.e., modeling the "real life" history and database history, respectively. Versioning and configuration management aspects of data, e.g. variants and alternatives, are not defined for temporal relational databases. These aspects are extensively covered by (temporal) object databases, see Skjellaug [61] for a survey on time in object databases.

Temporal models require that query languages are extended with temporal capabilities. One effect is that the set of objects handled by a language rises from that of only managing one relation kind to that of both managing temporal relations (relations with "times"), snapshot relations (relations without "times"), and lifespans (only "times"). With a temporal query language all the temporal information stored in a database may be manipulated

and explored through a query language by means of the built-in temporal semantics of the database system.

In summary, research in temporal relational databases has set the stage for several areas of research on temporal data, such as temporal object and engineering databases, time in active and deductive databases, multi-media databases, geographic information systems, scientific and statistic databases.

Bibliography

[1] T. Ahn, H. Jo, J. Kim, Y. Lee, and B. Kim. Graphic Interface for Temporal Summary Data (Extended Abstract). In *Proceedings of the 6th Korea and Japan Joint Conference on Statistics (invited paper)*, pages 3–8, 1989.

[2] C. H. Ang and K. P. Tan. The interval B-tree: a new time indexing technique. In *Proceedings of the 5th Australasian Conference*, pages 162–178, Christchurch, New Zealand, 1994.

[3] G. Ariav. A Temporally Oriented Data Model. *ACM Transactions on Database Systems*, 11(4):499–527, December 1986.

[4] G. Ariav, A. Beller, and H. L. Morgan. A Temporal Data Model. Technical Report DS-WP 82-12-05, Decision Sciences Department, University of Pennsylvania, December 1984.

[5] L. Baekgaard and L. Mark. Incremental computation of time-varying query expressions. *IEEE Transactions on Knowledge and Data Engineering*, 7(4):583–590, August 1995.

[6] J. Ben-Zvi. *The Time Relational Model.* PhD thesis, Computer Science Department, UCLA, 1982.

[7] J. van Benthem. *The Logic of Time.* Kluwer Academic Publishers, Dordrecht, Bosten, London, 2nd. edition, 1991.

[8] M. Böhlen, C. S. Jensen, and B. Skjellaug. Spatio-Temporal DataBase Support for Legacy Applications. Submitted for publication, April 1997.

[9] M. H. Böhlen and C. S. Jensen. Seamless Integration of Time into SQL. Technical Report R-96-49, Department of Computer Science, Aalborg University, 1996.

[10] S. Chaudhuri. Temporal Relationships in Databases. In *Proceedings of the International Conference on Very Large Data Bases*, pages 160–170, Los Angeles, California, 1988.

[11] J. Chomicki. Temporal Query Languages: A Survey. In Ohlbach H. J. Gabbay, D. M., editor, *Proceedings of the First International Conference on Temporal Logic*, pages 506–534. Lecture Notes in Artificial Intelligence 827, Springer-Verlag, July 1994.

[12] J. Clifford and A. Croker. The Historical Relational Data Model (HRDM) and Algebra Based on Lifespans. In *Proceedings of the International Conference on Data Engineering*, pages 528–537, Los Angeles, CA, February 1987. IEEE Computer Society Press.

[13] J. Clifford, C. Dyreson, T. Isakowitz, C. S. Jensen, and R. T. Snodgrass. On the Semantics of "Now" in Databases. *ACM Transactions on Database Systems*, 1997. Tentatively scheduled for Vol. 22, No. 1, March 1997.

[14] J. Clifford, C. Dyreson, T. Isakowitz, S. J. Jensen, and R. T. Snodgrass. *"Now"*, chapter 20. In R. T. Snodgrass (ed.) [66], 1995.

[15] J. Clifford and A. U. Tansel. On an Algebra for Historical Relational Databases: Two Views. In S. Navathe, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 247–265, Austin, TX, May 1985.

[16] J. Clifford and A. Tuzhlin, editors. *Recent Advances in Temporal Databases: Proceedings of the International Workshop on Temporal Databases*, Workshops in Computing, Zurich, Switzerland, September 1995. Spinger-Verlag.

[17] J. Clifford and D. S. Warren. Formal Semantics for Time in Databases. *ACM Transactions on Database Systems*, 8(2):214–254, June 1983.

[18] James Clifford and Tomas Isakowitz. On the Semantics of (Bi)Temporal Variable Databases. In *Proceedings of the International Conference on Extending Database Technology*, volume 779 of *Lecture Notes in Computer Science*, pages 215–230. Springer-Verlag, 1994.

[19] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, June 1970.

[20] P. Dadam, V. Lum, and H.-D. Werner. Integration of Time Versions into a Relational Database System. In U. Dayal, G. Schlageter, and L.H. Seng, editors, *Proceedings of the International Conference on Very Large Data Bases*, pages 509–522, Singapore, August 1984.

[21] C. De Castro, F. Grandi, and M. R. Scalas. On Schema Versioning in Temporal Databases. In Clifford and Tuzhlin [16], pages 272–291.

[22] R. Elmasri and V. Kouramajian. Indexing, Searching and Archiving Issues in Temporal Databases. In R. T. Snodgrass, editor, *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, Arlington, TX, June 1993.

[23] M. Finger. Handling Database Updates in Two-Dimensional Temporal Logic. *Journal of Applied Non-Classical Logics*, 2(2), 1992.

[24] S. K. Gadia. A Homogeneous Relational Model and Query Languages for Temporal Databases. *ACM Transactions on Database Systems*, 13(4):418–448, December 1988.

[25] S. K. Gadia. *Ben-Zvi's Pioneering Work in Relational Temporal Databases*, chapter 8, pages 202–207. In Tansel et al. [68], 1993.

[26] S. K. Gadia and C. S. Yeung. A Generalized Model for a Relational Temporal Database. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 251–259, Chicago, IL, June 1988.

[27] H. Gunadhi and A. Segev. A Framework For Query Optimization In Temporal Databases. In *Fifth International Conference on Statistical and Scientific Database Management Systems*, 1989.

[28] H. Gunadhi and A. Segev. Query Processing Algorithms for Temporal Intersection Joins. In *Proceedings of the 7th International Conference on Data Engineering*, Kobe, Japan, 1991.

[29] H. Gunadhi and A. Segev. Efficient Indexing Methods for Temporal Relations. *IEEE Transactions on Knowledge and Data Engineering*, 5(3):496–509, June 1993.

[30] C. S. Jensen. *Towards the Realization of Transaction Time Database Systems*. PhD thesis, Aalborg University, Department of Mathematics and Computer Science, Aalborg, Denmark, December 1990.

[31] C. S. Jensen, J. Clifford, R. Elmasri, S. K. Gadia, P. Hayes, and S. Jajodia [eds]. A Glossary of Temporal Database Concepts. *ACM SIGMOD Records*, 23(1):52–64, March 1994.

[32] C. S. Jensen and R. Snodgrass. Temporal Specialization and Generalization. *IEEE Transactions on Knowledge and Data Engineering*, 6(6):954–974, December 1994.

[33] C. S. Jensen, M. D. Soo, and R. T. Snodgrass. Unifying Temporal Models via a Conceptual Model. *Information Systems*, 19(7):513–547, October 1994.

[34] H. Kamp. Formal properties of 'now'. *Theoria*, 37(3):227–273, 1971.

[35] N. Kline. Bibliography containing entries relevant to Temporal DBMS's. http://liinwww.ira.uka.de/bibliography/Database/time.html, January 27, 1996.

[36] C. Kolovson. *Indexing Techniques for Historical Databases*, chapter 17, pages 418–432. In Tansel et al. [68], 1993.

[37] T. Y. Leung and R. Muntz. Generalized Data Stream Indexing and Temporal Query Processing. In *Second International Workshop on Research Issues in Data Engineering: Transaction and Query Processing*, February 1992.

[38] T.Y.C. Leung and H. Pirahesh. Querying Historical Data in IBM DB2 C/S DBMS Using Recursive SQL. In Clifford and Tuzhlin [16], pages 315–331.

[39] N. Lorentzos and R. Johnson. Extending Relational Algebra to Manipulate Temporal Data. *Information Systems*, 15(3):289–296, 1988.

[40] E. McKenzie. *An Algebraic Language for Query and Update of Temporal Databases*. PhD thesis, Computer Science Department, Univ. of North Carolina at Chapel Hill, September 1988.

[41] E. McKenzie and R. Snodgrass. Schema Evolution and the Relational Algebra. *Information Systems*, 15(2):207–232, June 1990.

[42] E. McKenzie and R. Snodgrass. An Evaluation of Relational Algebras Incorporating the Time Dimension in Databases. *ACM Computing Surveys*, 23(4):501–543, December 1991.

[43] E. McKenzie and R. T. Snodgrass. Supporting Valid Time in an Historical Relational Algebra: Proofs and Extensions. Technical Report TR–91–15, Department of Computer Science, University of Arizona, Tucson, AZ, August 1991.

[44] J. Melton and A.R. Simon. *Understanding the New SQL: A Complete Guide*. San Mateo, CA: Morgan Kaufmann Publishers, Inc., 1993.

[45] J. Melton (ed.). Database Language SQL. ISO/IEC JTC 1/SC21 WG 3 DBL:RIO-004 (ANSI TC X3H2-94-329), August 1994.

[46] J. Melton (ed.). Framework for SQL. ISO/IEC JTC 1/SC21 WG 3 DBL:RIO-003 (ANSI TC X3H2-94-328), August 1994.

[47] R. Montague. *Formal Philosophy: Selected Papers of Richard Montague*. Yale University Press, New Haven, 1974.

[48] S. B. Navathe and R. Ahmed. A Temporal Relational Model and a Query Language. *Information Sciences*, 49:147–175, 1989.

[49] P. Øhrstrøm and P. F. V. Hasle. *Temporal Logic: From Ancient Ideas to Artificial Intelligence*. Kluwer Academic Publishers, 1995.

[50] G. Özsoyoğlu and R. Snodgrass. Temporal and Real-Time Databases: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 7(4):513–532, August 1995.

[51] N. Pissinou, R. T. Snodgrass, R. Elmasri, I.S. Mumick, M.T. Özsu, B. Pernici, A. Segev, and B. Theodoulidis. Towards an Infrastructure for Temporal Databases: Report of an Invitational ARPA/NSF Workshop. *ACM SIGMOD Records*, 23(1):35–51, March 1994.

[52] A. Prior. *Past, Present and Future*. Oxford University Press, 1967.

[53] A. N. Prior. *Time and Modality*. Clarendon Press, Oxford, 1957.

[54] N. C. Rescher and A. Urquhart. *Temporal Logic*. Springer-Verlag, New York, 1971.

[55] J. F. Roddick. SQL/SE - A Query Language Extension for Databases Supporting Schema Evolution. *ACM SIGMOD Records*, 21(3):10–16, September 1992.

[56] J. K. Roddick and R. T. Snodgrass. *Schema Versioning Support*, chapter 22, pages 427–449. In R. T. Snodgrass (ed.) [66], 1995.

[57] R. v. B. Rucker. *Geometry, Relativity and The Fourth Dimension*. Dover Publications, Inc., New York, 1977.

[58] B. Salzberg. On Indexing Spatial and Temporal Data. *Information Systems*, 19(6):447–465, 1994.

[59] N. Sarda. Algebra and Query Language for a Historical Data Model. *The Computer Journal*, 33(1):11–18, February 1990.

[60] A. Segev, C. S. Jensen, and R. T. Snodgrass. Report on The 1995 International Workshop on Temporal Databases. *ACM SIGMOD Records*, 24(4), December 1995.

[61] B. Skjellaug. Temporal Data: Time and Object Databases. Research Report 245, Department of Informatics, University of Oslo, April 1997. ISBN 82-7368-160-2.

[62] R. Snodgrass, editor. *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, Arlington, TX, June 1993.

[63] R. T. Snodgrass. The Temporal Query Language TQuel. *ACM Transactions on Database Systems*, 12(2):247–298, June 1987.

[64] R. T. Snodgrass, M. H. Böhlen, C. S. Jensen, and A. Steiner. Adding Transaction Time to SQL/Temporal. ANSI Expert's Contribution, ANSI X3H2-96-502r2, ISO/IEC JTC1/ SC21/WG3 DBL-MAD-147r2, International Standards Organization, November 1996.

[65] R. T. Snodgrass, M. H. Böhlen, C. S. Jensen, and A. Steiner. Adding Valid Time to SQL/Temporal. ANSI Expert's Contribution, ANSI X3H2-96-501r2, ISO/IEC JTC1/ SC21/WG3 DBL-MAD-146r2, International Standards Organization, November 1996.

[66] R. T. Snodgrass (editor). *The TSQL2 Temporal Query Language.* Kluwer Academic Publishers, 1995.

[67] A. U. Tansel. Adding Time Dimension to Relational Model and Extending Relational Algebra. *Information Systems*, 11(4):343–355, 1986.

[68] A. U. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass (eds.). *Temporal Databases: Theory, Design, and Implementation.* Database Systems and Applications Series. Benjamin/Cummings, Redwood City, CA, 1993.

[69] D. Toman. Point vs. Interval-based Query Languages for Temporal Databases. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART PODS*, Montreal, Canada, June 1996.

[70] V. J. Tsotras and A. Kumar. An Update of the Temporal Database Bibliography. *ACM SIGMOD Records*, 25(1), March 1996.

[71] C. Vassilakis, N. Lorentzos, and P. Geogiadis. Transaction Support in a Temporal DBMS. In Clifford and Tuzhlin [16], pages 255–271.

[72] G. Wiederhold, S. Jajodia, and W. Litwin. Dealing with Granularity of Time in Temporal Databases. In *Proc. 3rd Nordic Conf. on Advanced Information Systems Engineering*, Trondheim, Norway, May 1991.