A Critical Review of Temporal Database Management Systems

By

Fang, Weiqi B.Sc.

A Dissertation Submitted in Fulfilment of the Requirements

for

the Degree of Master of Science

in the Department of Computing Science

at the University of Glasgow

April 1989

To my motherland

Acknowledgments

I would like first of all to express my gratitude to my supervisor Ray Welland, for introducing me to the excitement of Temporal Databases as an inter-disciplinary research area; for his many helpful discussions that contributed to clarifying the ideas in this dissertation and their presentation herein and his encouragement in my study.

I am greatly indebted to Professor H.Y. Wong (Professor of Aeronautics & Fluid Mechanics). He suggested and organized a Special Training Programme for a group of Guangdong students from China. After that, I became a student in Glasgow University. His keen interest and his thoughtful comments on our group brought progresses to my postgraduate study.

My deep thanks should be expressed to the Shantou Government in China for their financial support and the opportunity they gave me to study for Double Master Degrees in Scotland. Professor D.C. Gilles and Professor M.P. Atkinson deserve credit for accepting me to study in the department of Computing Science.

Dr. Jeacocke pointed out the problem of what a tuple represents in a temporal database. This emphasizes the need to discuss the concepts of entity, entity state, observation of entity, and observation of entity state in depth in this dissertation. I am grateful to Professor Snodgrass and his colleagues for their contributions on TDB and TQuel, which have been taken as a basic model in my study; and to my colleagues for insightful observations that greatly improved the dissertation.

Finally, I wish to thank my family, especially my wife, and all my friends here and in China for their help, understanding, assistance and encouragement.

Contents

Summary

There have been significant research activities in *Temporal Databases* during the last decade. However, the developments of a semantics of time, a temporal model for efficient database systems and temporal query languages still need much study.

Based on the researches of the TDB group [Snodgrass 1987], the review of research about TDBMS in this dissertation mainly emphasises three aspects as follows.

1) *The formulation of a semantics of time at the conceptual level.* A topology of time and types of time attributes are introduced. A new taxonomy for time attributes is presented: assertion time, event time, and recording time.

2) *The development of a model for TDBMS analogous to relational databases.* Based on Snodgrass' classification, four kinds of databases: snapshot, rollback, historical and temporal are discussed in depth. But the discussion distinguishes some important differences from the representation of the TDB model:

- historical relation for most enterprises is an interval relation, but not a sequence of snapshot slices indexed by valid time.
- the term "tuple" no longer simply refers to an entity as in traditional relational databases. It refers to different level representations of an object: entity, entity state, observation of entity, and observation of entity state in different types of databases.

3) *The design of temporal query languages.* We do not present a new temporal query language in this dissertation, but we discuss a Quel-like temporal query language, TQuel, in some depth. TQuel is compared with two other temporal query languages TOSQL and Legol 2.0. We centre the main discussion on TQuel's semantics for tuple calculus. The classification for the relationships between overlapping intervals suggests an approach using temporal logic to classify the derived tuples in tuple calculus. Under such an approach, a new presentation for tuple modification calculus is proposed, not only for interval relations, but also for event relations.

Chapter 1 Introduction

1.1 The Role of Time

Information about the constantly evolving real world need to be interpreted in the context of time. Causal relationships among events or entities are embedded in the temporal information. For example, since the early age of human beings, the birth date has been taken as an attribute to stamp a man when he was born (valid) in the world. This attribute seems to be the first time stamp attribute of data (personnel) in information processing. In most information management applications time is a universal attribute and deserves special treatment as such.

With the development of information processing, the problem of representing the time aspect of information arises in a wide range of disciplines, specially, in computer science, philosophy, temporal logic and linguistics. In computer science, it is a core problem of information system modelling, software engineering, artificial intelligence, distributed systems, and other areas involving data structure.

In information systems, for instance, the traditional approach to deal with the problem of outdated data is simply to delete it; however, this eliminates the possibility of accessing any information which is not presently current. In order to consider queries such as, "Which students were members of the library last year and borrowed over twenty books," we need to represent temporal information. In some applications, such as booking tickets and making appointments, the time course of events becomes a critical part of data. In artificial intelligence, models of problem solving require sophisticated world models that can capture change. Making decisions in a banking system, for instance, one must model the effects of the currency trends to ensure that a decision will be effective. In natural language processing, extracting and capturing temporal and tense information in sentences are necessary. Temporal knowledge is necessary to be able to

answer queries about the sentences later. Further progress in these areas requires more powerful representations of temporal knowledge than have been available previously.

1.2 Time in Databases

Database technology plays an important role in all of these areas to present a good mechanism to record information. *Temporal database Management Systems* are an important research area studying how to represent time aspects in database management systems.

"Databases supposedly model reality, but conventional database management systems (DBMSs) lack the capability to record and process time-varying aspects of the real world. With increasing sophistication of DBMS applications, the lack of temporal support raises serious problems in many cases. For example, conventional DBMSs cannot support historical queries about past status, let alone trend analysis (essential for applications like decision support systems). There is no way to represent retroactive or proactive changes, while support for error correction or audit trail necessitates costly maintenance of backups, checkpoints, or transaction logs to preserve past states." [Snodgrass & Ahn 1986] In general, none of the traditional data models (relational, network, hierarchic databases and so on) explicitly addresses temporal or historical aspects of the data.

In the practical realm of information systems, time aspects are usually either neglected, treated only implicitly, or explicitly factored out, in spite of the abundance of temporal references in common data. As a result, most information systems and generalized data management tools do not treat "present" data and "past" data symmetrically, but typically differentiate between them in terms of accessibility - both logical and physical. The conventional database in the core of many information systems is a "thin", tenseless, and temporally inconsistent snapshot of latest available data. Data items within an instance of such a database pertain to various points in time, newly recorded data eventually replaces previously recorded ones. If kept at all, "forgetfully"

replaced values are usually retained on "log" files, meant mainly to allow recovery of damaged data. All these practices imply substantial limitations on the range and economic feasibility of historical inquiries and "what if" analyses that information systems can support.

On the other hand, traditional databases also lack the capability to present the evolution of future objects. A future object is not really the same as a historical object in that the object in the future is not yet a reality. Some instances may come to be true and some may not; one has to separate the future instances from those which are current and those which form the history chain. Those future instances (projected events) which "happen" will become current and eventually pass into the history chain. However, those future instances which do not happen will have to move into a different history chain, or be maintained in some special way.

Obviously, in order to retain complete information about an object, the current data of its attributes, as well as their histories and future trends, should be stored and managed by the DBMS.

Time logically adds another dimension to a data model. The concept of time is crucial to all databases, but is only treated implicitly in the existing database models. Many applications have been forced to manage temporal information in an ad-hoc manner. For example, time is simply modelled as an attribute in such databases. In a library model [Oxborrow 1986], for instance, date due back and date reserved are both time-oriented objects which are modelled as attributes. In such an approach, first, the effect of time is completely hidden in the "current view" model. Under this circumstance, much of the information about events that have occurred is not available. For example, we cannot record the activities of renewing books with such a model. After an update has been made we do not know whether it was a new attribute value assumed by an object or it was a correction. All we can do is either to ignore the requirement of renewing, or to forget (delete) the historical data of last borrowing activity, or to keep two tuples with the same candidate key values causing confusion about which tuple is presently true.

Secondly, deletion, in the model, means that the data is modified and the values from that time on will be nonexistent. Therefore, the integrity of time-oriented objects cannot be guaranteed. In addition, the two time attributes are not candidate keys (candidate keys may be the book number and so on), but one must include the time aspect with the candidate keys in order to uniquely identify the tuple being addressed. We have to manage temporal information in an ad-hoc manner. While such an approach may have been satisfactory in some cases, its success is limited and its use is restrictive. What is used in one application may have to be implemented again in another.

Databases exist in time and model changes that occur temporally in the world via database state changes. In order to have a proper understanding of how an explicit representation of time interacts with all of the data in the database, it is not enough simply to allow users to utilize "time attributes" where they seem appropriate. It is necessary to embed time aspects into the system level in databases, both at the conceptual level and internal level. By incorporating general temporal semantics directly within the database model, not only do we spare the user the task of defining such a semantics, but we also can ensure that time is treated in a uniform and consistent manner. Moreover, if the temporal semantics are built into the model, implementations of a temporal database can take advantage of this standard semantics to increase the efficiency of database operations.

The need for providing temporal support in DBMS has been recognized since the last decade [Bubenko 1977]. There have been significant research activities in formulating a semantics of time at the conceptual level, developing a model for time varying databases analogous to the relational model for static databases [Clifford & Warren 1983, Codd 1979], and the design of temporal query languages [Ariav & Morgan 1981, Snodgrass 1982]. In particular, a working group of TDB (Temporal DataBases), led by Professor Snodgrass, in the Department of Computer Science at the University of North Carolina has developed a model of temporal databases and its language TQuel - an extension of the database language of INGRES since that time. They presented a new

taxonomy of three distinct time concepts (termed as *transaction time*, *valid time*, and *user-defined time*), and four distinct kinds of database(*snapshot, rollback, historical,* and *temporal*), differing in their support of the new time concepts.

1.3 Dissertation Plan

Based on the researches of TDB group, the discussion in this dissertation will focus on a relational temporal database model and the semantics for its language, and thus can be mainly divided into two parts:

1.3.1 The basic model

There are two chapters discussing the topology of time, taxonomy of time attributes and database models.

1) In Chapter 2, the basic concepts of time will be introduced. A linear, step-wise constant and semi-closed time model is discussed. To fully capture time-varying information, as the information environment is being classified into user level, event level, and system level, time attributes will be classified into three types: assertion time, event time and recording time which is a different taxonomy of time attributes from that of researchers.

2) In Chapter 3, differentiated by their ability to represent temporal information, four types of databases are introduced:

- snapshot databases -- representing the current content of data only,
- rollback databases -- representing the history of database system activity,
- historical databases -- representing the history of real world,
- temporal databases -- providing support for representing the enterprise being modelled and the history of database activities at the same time and fully capture the history of retroactive and proactive changes of data.

To represent the enterprise modelled in the database, a new taxonomy of entity, entity state, observation of entity, and observation of entity state will be presented. This classification clarifies the different concepts between tuple and entity in the four distinct database types and would be useful to capture more temporal meaning of entities in the real world.

1.3.2 The language semantics

We will not design a new temporal language, but will discuss almost all semantics statements in TQuel. Chapter 4 will present the background material for TDB and comparison of TQuel with another two temporal languages: TOSQL and Legol 2.0. The discussion shows that it is worth taking TDB and TQuel as the central model in this dissertation.

To correct semantic problems in TQuel, specially in modification statements, an approach which proposes a temporal logical classification for the relationships between overlapping intervals will be introduced. The basic predicate *Before* will be modified as *less than*, but not *less than or equal to* to avoid the semantics problem as well. Chapter 5 is the main chapter to describe such modifications.

Chapter 2 The Basic Concepts of Time

It is necessary to distinguish the basic points of view about time concepts before discussing how to embed time into databases. The most important concepts are the topology of time and the types of time attributes in temporal databases. In this chapter both concepts will be discussed. The discussion is based on reviewing previous characterizations presented by many researchers, specially on the work on TDB and TQuel by Snodgrass et al [Snodgrass 1987, Ahn & Snodgrass 1986]. We shall argue that the types of time attributes should be classified into a new taxonomy with three time concepts: *assertion time*, *event time* and *recording time* and that these concepts differ in the level at which they are applied:

~ the user level,

~ the event level, or

~ the system level.

2.1 Topology of Time

The means to express and explore the consequences of the structure of time is very different in different studies of Temporal DataBase Management Systems (TDBMS). These differences influence:

~ the structure modelling of TDBM, i.e., temporal schema creation;

~ the temporal algebra;

~ the semantics and syntax of TDBM languages.

Several topologies of time which have been proposed are now discussed.

2.1.1 Interval time vs. event time

Time can be referred as points or intervals. We say that, for instance, the quantity of new books in the library increased by the amount of 400 in this October. We consider

the fact of quantity increasing happened at the time point of this October. If we say that the quantity of new books in the library has increased by the amount 400 during this October, then the time, October, is considered as a time interval. We define time point as event time and time interval as interval time.

Representing time as a point is simple and requires less storage space. However, to determine the time duration over which a value is valid, the successor pair has to be examined. This creates complications in expressing and interpreting the algebra operations [Clifford & Tansel 1985, Allen 1983].

The representation of an event in most approaches we have studied is a tuple that exists for exactly one valid time, with the snapshot slices of the previous and next valid times not containing the tuple. This representation is problematic because time is continuous: It is misleading to talk about the previous and next time values. Therefore, only states that exist for a finite interval of time may be represented, while events, occurring instantaneously, are more difficult to model.

There are examples which provide counter-intuitive results if we allow zero-width time points. For instance, consider the situation where a light is turned on. To describe the world changing we need to have an interval of time during which the light was off, followed by an interval during which it was on. The question arises as to whether these intervals are open or closed. If they are open, then there exists a time (point) between the two where the light is neither on nor off. Such a situation would provide serious semantic difficulties in a temporal logic. On the other hand, if the interval is closed, then there is a time point at which the light is both on and off. This presents even more semantic difficulties than the former case. One solution to this would be to adopt a convention that intervals are closed in their lower end and open on their upper end[1]. The intervals could then meet as required, but each interval would have only one end point. The artificiality of this solution merely emphasizes that a model of time based on points on the real line does not really correspond to our intuitive notion of time.

[1] See Section 2.1.3 Bounded or unbounded.

Most temporal databases only support event time or interval time, but Snodgrass' TDBMS model supports both of them in valid time and user-defined time attributes (see Chapter 4 Some Proposed Temporal Databases and Their Languages)). However, the model only supports event time for each transaction event.

We summarize some researches which support event time, or interval time, or both of them in Figure 2.1.

Reference	Model	Event time	Interval time
[Findler & Chen 1971]	AMPPL-II	√	√
[Bubenko 1977]	--	√	√
[Reed 1978]	SWALLOW		√
[Jones & Mason 1980]	Legol 2.0		√
[Klopprogge 1983]	TERM	√	
[Ben-Zvi 1982]	TRM		√
[Clifford & Warren 1983]	ILs	√	
[Lum et al. 1984]	AIM	√	
[Clifford &Tansel 1985]	HDBM [1]/HDB [2]	√	√
[Snodgrass & Ahn 1985]	TDB	√	√
[Ariav 1986]	TODM (TOSQL)	√	
[Gadia & Yeung 1988]	MHM		√

Figure 2.1 Event Time and Interval Time

2.1.2 Discrete or continuous

Time is universal continuous and can be treated as dense, essentially isomorphic to the set of reals. However, we can consider it as being discrete. For example, if one is dealing with facts which only vary on a time scale consisting of whole numbers of days, then one can take a day as an indivisible unit and use a discrete time model. Again, suppose we have *n* property-variables, each of which is present or absent at any one time, and that we define an *epoch* to be a maximal period over which none of the variables changes value. Then the succession of epochs constitutes a discrete time

[1] Clifford's model. It supports event time.
[2] Tansel's model. It supports interval time.

structure[1] imposed on what may very well be an intrinsically continuous underlying temporal model.

For two reasons, we prefer to treat time as discrete, and isomorphic to the natural numbers. First, it is clear that any recording instrument must have at best a finite sampling unit, and second, any practical language that we might define for time attributes in a TDBM would have at most a countably infinite set of names for time points or time intervals. Thus while it may be philosophically or theoretically interesting to consider a continuum of moments of time, from a practical standpoint the natural numbers seem a more useful candidate for modelling the properties of database time.

2.1.3 Bounded or unbounded

Is time bounded or unbounded? This means that either every time is succeeded by a later time (unbounded in the future) or there is a last moment in time (bounded in the future), and likewise in the direction of the past. As stated in Section 2.1.1, if we present the time as interval time and the interval is closed (may be bounded by a later time), then there is a time point at which two events both exist or do not exist. Under such a model, the time intersection operation may generate some redundant tuples which have the same lower end and upper end (see Chapter 5 New Semantics for TQuel Modification Statements). One solution to this would be to adopt a convention that intervals are closed in their lower and open on their upper end, i.e., are unbounded (or semi-closed).

2.1.4 Linear, parallel, or branching

Generally, time is considered as a linear order, because in Newtonian physics time is modelled as a real line. The relationship of two time points in the historical chain can be expressed as a predicate "Before", (or a notation "<"). In such an approach, the predicate "Before" is restricted to be transitive and connected.

---

[1] Some papers defined such a topology of time as step-wise constant type (see [Segev & Shoshani 1987]).

However, the idea of branching time has been put forward as a way of handling uncertainty about the future, the idea being that from each moment forward there exist many possible alternative futures. For instance, those future instances which 'happen' will become current and eventually pass into the history chain. And those future instances which do not happen will have to move into a different history chain, or be maintained in some special way. The time of such instances have a branching structure. Consider the following example: during 1980 to 1988, the department D1 of a store sold an item I2 . If the item I2 was supplied by company C1 from 1980 to 1988, and additionally by company C2 from 1985 to 1988, then the time of the event: The department D1 sold item I2, has a branching time structure to distinguish different suppliers.

As for parallel times: we could perhaps regard this as a model of the different subjective time-scales of different people, but again, it would seem necessary to establish correlations between the different times, and the robustness of such correlations is a measure of how well-founded is the idea of an objective time underlying all the different subjective time-lines.

Circular time is an interesting possibility, too, for in this, past, present and future coalesce. It is hard to take circular time seriously as an account of physical time, but the associated logic could be useful in reasoning about repetitive processes, for instance the effectively endless repetition of cycles in the traffic signals at a road junction.

In the following discussions, the TDB presented by Snodgrass et al. is based on a topological structure of time which is linear, step-wise constant and semi-closed (unbounded), and supports both event time and interval time as well.

2.2 Different Types of Time Attributes

So far, many researchers have proposed a variety of time terminologies. They have treated time aspects of conceptual information models in different ways, argued distinct taxonomies of time attributes, then presented different types of TDBMS (Temporal Database Management System) models. In discussing these concepts to

examine their contributions to temporal database research, we introduce a new classification of time attributes. In the new taxonomy of time, two time attributes are defined as almost the same as *logical* time and *physical* time which were presented by Lum et al. in 1983. However, we shall point out that another time attribute is also necessary to fully capture time-varying information. These three new time attributes are different from the classification of time attributes by Snodgrass et al. [Snodgrass & Ahn 1985] as well. We shall proceed by stating different viewpoints presented by major authors, then follow with an analysis of these views, comparing their differences and linking their common characteristics.

2.2.1 The different viewpoints of researchers

2.2.1.1 Rescher and Urquhart's views

In [Rescher & Urquhart 1971], the authors stated that there were two time attributes associated with a statement which describes a situation or a state of affairs. The first attribute is the time of *reference* of a statement which expresses when a particular condition exists or an event happens. The second attribute is the time of *assertion* of the statement which expresses when the statement is asserted or uttered by someone. Often the reference time *depends on* the assertion time. As in the statement, "it is now raining", which was issued on 15/10/88, the assertion time is "15/10/88", and the reference time is "now" which refers to "15/10/88". If we issued the statement, "it was raining yesterday", on last Sunday, then the assertion time is "last Sunday", and the reference time is "yesterday" and is relative to "last Sunday", i.e., is last Saturday.

2.2.1.2 Bubenko's views

In an abstract model realm in [Bubenko 1977], time also plays two kinds of roles: *extrinsic* and *intrinsic*. The extrinsic time is the time when a particular assertion is made or conclusion is drawn. Whether this time relationship is explicitly recognized or not *it*

always exists. However, different applications have different views of this dimension. Therefore, extrinsic time is application-specific. Intrinsic time plays a role as part of the definition of an association type, i.e., it constitutes parts of its "meaning". An association type may or may not contain intrinsic time components.

Consider the following examples: we draw a conclusion on 15/10/88 (t1): "The quantity on hand of ACM magazines is 8". Here, "15/10/88" (t1) is an extrinsic time. Next, consider the assertion on 25/10/88 (t2) about the above conclusion "On 15/10/88 the quantity on hand of ACM magazines was 8". Observe that the latter is true for all extrinsic t2 > t1 and that t1 has now moved to an intrinsic role, i.e., "15/10/88" now is an intrinsic time and the extrinsic time is "25/10/88". Consequently, t2 will also move to an intrinsic role if we assert at t3 > t2 the assertion about the first conclusion etc. That means that an extrinsic time can move to an intrinsic role with the development of history. In most applications, we will normally have events which depend on the extrinsic time. In some applications there are some events which do not depend on time, these represent "static data" where intrinsic time relation implicitly or explicitly always is present. For example, we said "he is a man" last Monday. We still draw the same conclusion this Monday, "he is a man". We never say, "he was a man last Monday". The statement, "he is a man", is static. Clearly, extrinsic time is assertion time and intrinsic time is reference time in [Rescher & Urquhart 1971].

2.2.1.3 Lum's views

In [Lum et al. 1984], the concepts of time were classified into two categories, *physical* time and *logical* time. Physical time is characterized as the time when the data concerning the event was stored in the database, while logical time is characterized as the time that an event occurs in reality. The physical time is used to record all database actions. It is running on a non-stop running clock (e.g., system calendar), and cannot be modified by users at all. Some literature named such a time attribute as transaction time [Snodgrass & Ahn 1985], or recording time [Ariav 1986]. Physical time is automatically

processed by DBMS. It is taken as a time stamp embedded within each transaction in a database. Considering only single user applications, physical time is in linear order and its integrity can be guaranteed by DBMS.

While physical time is necessary to be the reference point, the authors argued that there was another aspect of time that must be considered. That is, the time associated with the user's application perspective. They referred to this time aspect as logical time. Logical time concerns modelling time-varying reality. It states when an event became effective. And it is considered as a second reference point in time based on the real physical time as a reference. It can be defined by users explicitly. Therefore, it is application dependent and its integrity must be maintained by the user. For any system, only *one* physical time attribute can exist, while there may be *several* logical time attributes. Comparing the definitions in [Bubenko 1977, Rescher & Urquhart 1971], logical time is *intrinsic* time or *reference* time. In [Ariav 1986], logical time is observation time.

2.2.1.4 Snodgrass and Ahn's views

In [Snodgrass & Ahn 1985, Snodgrass & Ahn 1986], the authors identified three kinds of time that a database management system needs to support: *valid* time, *transaction* time and *user-defined* time. The new definition was based on reality versus representation.

The transaction time of an event is the transaction number (an integer) of the transaction that stored the information concerning the event in the database. It concerns the storage of information in the database. Such a time can be considered as a daily calendar. When the data are recorded into databases, this time attribute will be embedded, as with a time stamp. It may not be changed. Therefore, a database dealing only with transaction time will permit only appends, no modifications will be allowed. The transaction time is automatically processed by TDBMS. If the value can be processed automatically by the DBMS, the value must necessarily be independent of any particular

application and must have a simple semantics. Hence, the transaction time must be application independent.

The valid time of an event is the clock time that the event occurred in the world, independent of the recording of that event in some databases. A database concerned only with the valid time permits modifications, because valid time concerns modelling time-varying reality and is always subject to change. However, the authors argued that valid time is application-independent.

User-defined time is an uninterpreted domain for which the DBMS supports the operations of input, output, and perhaps comparison and minimal computation. Such domains will be present in the relation schema. The values of user-defined temporal domains are not interpreted by the DBMS. The semantics of user-defined time is provided by the user or application program.

Transaction time is most closely associated with physical time while valid and user-defined times are associated with logical time.

2.2.2 Discussions

2.2.2.1 The valid time and user-defined time

In [Snodgrass & Ahn 1985], the authors presented a new taxonomy of time by arguing that there has been some confusion concerning terminology and the definition of physical time and logical time attributes presented in [Lum et al. 1984]. The *user-defined* time differs from *valid* time only because it is application dependent. According to the definition, user-defined time is application dependent, while valid time is application independent. But both of them are most closely associated with logical time. Unfortunately, it should be pointed out that valid time is also application dependent in most applications. We cannot separate valid time from user-defined time. As an example, consider the promotion relation shown in Figure 2.2 from [Snodgrass & Ahn 1985].

Name	Rank	effective date	valid time (at)	transaction time (start)	(end)
Merrie	associate	01/09/77	25/08/77	25/08/77	∞
Merrie	full	01/12/82	11/12/82	15/12/82	∞
Tom	full	05/12/82	05/12/82	01/12/82	07/12/82
Tom	associate	05/12/82	07/12/82	07/12/82	∞
Mike	assistant	01/01/83	01/01/83	01/01/83	∞
Mike	left	01/03/84	25/02/84	25/02/84	∞

Figure 2.2 A Promotion Example

Merrie's retroactive promotion to full (professor) was signed four days before it was recorded in the database. Tom in the full rank was recorded four days before it was valid. And a correction was made to this event seven days after: Tom should have had the rank of associate.

The effective date is the date shown on the promotion letter that the promotion was to take effect; the valid date is the date the promotion was signed, i.e., the date the promotion was validated; and the transaction date is the date the information concerning the promotion was stored in the database. The authors argued that the effective date is application-specific while the valid time should be application independent. However, consider that the letter must be signed by many authoritative persons, the letter may not be signed within one day. There should be multiple *valid at time* attributes to fully capture the history of promotion. It is totally dependent on the application how many valid at time attributes are enough. Therefore, the valid time is application dependent as well.

On the other hand, the valid time attributes can be defined and specified by the user, e.g., retrieved by issuing the **when** clause and modified by using the **valid** clause in TDBMS (this will be discussed in Chapter 4 in more detail). Because the valid time can be modified by users, the TDBMS cannot guarantee the integrity of valid time values.

Therefore, the valid time attribute cannot be considered as being application independent. In fact, the valid time is a kind of user-defined time. We classify valid time with user-defined time into the same taxonomy, logical time. All of them are characterized as the time that an event occurs in reality.

2.2.2.2 Logical time

As stated above, physical time and logical time classify two different concepts of time. Physical time states the concept of time from the viewpoint of machines (at the internal level of system). On the other hand, logical time views the concept of time of an object from an external level. Unfortunately, there has been some confusion concerning the viewpoint from external level. In a database system, we can view an object from an internal level of system or an external level. At external level, there still are two aspects of the object. One is viewing the object from the user points of view. Another is from the viewpoint of the object itself. As will be stated below, time also can be viewed from these two distinct aspects which are independent of each other.

2.2.2.3 Assertion time vs reference time and physical time

Both assertion time and reference time view time concepts from the external level of system as well. The *assertion time* is the time when someone at some point in time asserts or utters a statement which states the fact of an event happening. It concerns a different level of time concept from the reference time. The concept of reference time associates the time with an event itself, while the concept of assertion time differs from reference time in the fact that it concerns the time of observation by someone, although assertion time can become reference time with history changing. Assertion time really views time concepts of an object from the user level and reference time views time concepts from an event view. We term *reference time* as *event time* to make the terminology clear. Assertion time is also different from physical time with the fact that

physical time views the concepts of time from the system level, not from the external level. We term *physical time* as *recording time* to make the terminology clear as well.

2.2.3 The new classification

Now, we can redefine three orthogonal time concepts as follows to classify the types of time attributes for temporal database systems:

The **assertion time** is the time when someone at some point in time asserts or utters a statement which states the fact of an event happening. It views the concept of time of an object from the user level.

The **event time** is the time that an event occurs in reality. It states the aspect of time from the event level. With the development of history, assertion time can be changed into event time.

Assertion time depends on different users and different applications, while event time depends on different applications but does not depend on users. Both of them are application dependent. Assertion time tends to be changed at any time by relative users, while event time is changeable in future relations, but unchangeable in historical relations. This is due to the fact that an event, once it occurs, never becomes false. However, because events are recorded into DBMS by users and recording errors may be generated, events which have been recorded into the system should be allowed to be modified and event time can also be replaced in historical relations.

The **recording time** is characterized as the time when the data concerning the event was stored in the database. It views the time concept of an object from the system level. There will be not any recording time before the event is recorded into the TDBMS. After an event is recorded into the system, it is stamped with a recording time which cannot be changed at all. Therefore, recording time does not exist in the future chain and is not allowed to be changed in the present and the past by any user. It is automatically maintained by TDBMS after each transaction.

According to such definitions, a new classification can be drawn as Figure 2.3.

Concepts of Time	Logical time		Physical time
Terminology / Reference	Assertion time	Event time	Recording time
[Rescher & Urquhart 1971]	assertion time	reference time	
[Findler & Chen 1971]		start/finish time	
[Bubenko 1977]	extrinsic time	intrinsic time	
[Reed 1978]			start/end time
[Jones & Mason 1980]		start/end time	
[Ben-Zvi 1982]		effective time	registration time
[Clifford & Warren 1983]		state	
[Mueller & Steinbauer 1983]			data-valid time
[Lum et al. 1984]	logical time	logical time	physical time
[Copeland & Maier 1984]		event time	transaction time
[Clifford &Tansel 1985]			time[1]
[Snodgrass & Ahn 1985]	user-defined time	valid time / user-defined time	transaction time
[Ariav 1986]	observation time		recording time

Figure 2.3 A Classification of Time Concepts.

2.2.4 A tree-like hierarchy of time

For most objects, there is not just one aspect of time with them, but many aspects. Time is multi-dimensional. Different points of view associate different time attributes (time dimensions) with one object. Although we only classify time concepts into three aspects, they can be developed into multiple time attributes. The result is that time has a tree-like hierarchy.

For instance, time can be divided into logical time (external level view) and physical time (internal level view). The external level view of time also can be classified at two further levels: user level and event level. Assertion time and event time can be defined as above. For the same object, in different applications, different event times can

[1] This paper was a combination of two individual papers. One was written by James Clifford, and the other by Abdullan Uz Tansel. Here, we present Tansel's viewpoint. Clifford's viewpoint is as the same as stated in [Clifford & Warren 1983].

be obtained. Different users and different applications can view assertion time in different ways. We summarize the time concepts into a simple diagram as shown in Figure 2.4:

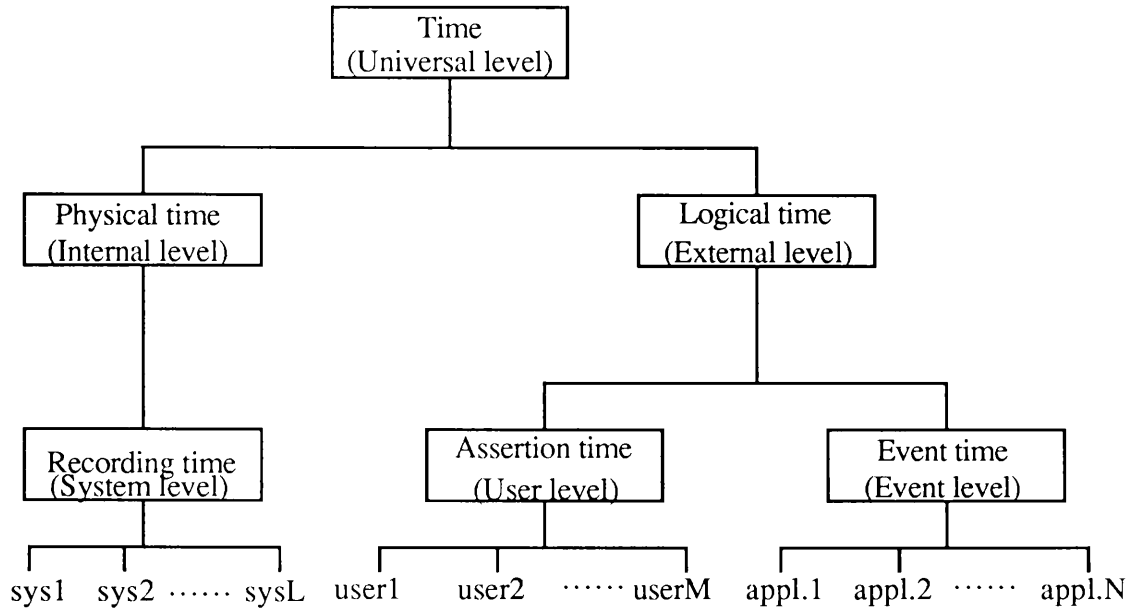


Figure 2.4 The Tree-like Hierarchy of Time

Some examples will help clarify the subtle differences among the three types of time attributes above and highlight their similarities:

1) Promotion example.

For promotion event, we can assert that the event happens to somebody at different views. We can assert the statement at the time when the promotion is discussed, or at the time when we make an agreement for the promotion, or at the time when the promotion document is signed. Therefore, three assertion time attributes can be obtained for the same event by the same user (but different applications).

There would be a promotion effective time as the event time. Such a time attribute may refer to any one of three assertion times above. We also can consider promotion activity in different applications. If we focus on the discussion of promotion, for instance, we have promotion discussion event time. If we focus on promotion agreement or signature of promotion, we have promotion agreement event time, or promotion

signature event time. Clearly, for the same object (promotion), in different applications, we have different event times. At the same time, we can see that assertion times can be evolved into event times in the history chain.

For such an example, obviously, the recording time is the time when an event is recorded into the system.

2) Birthday example.

The date a person was born is the birthday of that person. It is the event time of the event that the person comes into being. While the date when a certificate is signed to certify the valid being for that person is the assertion time for such an event. The assertion time can be taken as a reference time for the event time. The recording time is the time when such information is recorded into an information system.

3) Deposit example

A customer deposits some money at a branch of a large bank (event). The branch transmits the details of the payment to the central bank (assertion) at a later time, and finally the payment is recorded in a central computer (recording). The deposit transaction therefore has three times associated with it, but the bank's computer system may only record the final one. This makes it difficult for the customer to reconcile his deposit event with the bank's recording of this event.

4) Business transaction example

Thinking about the customer's point of view, he writes a cheque and sends it to a supplier of goods (payment assertion). The cheque is sent to the supplier's bank several days later by the supplier and money is really transferred from the customer's account to the supplier's one (payment event happens). The recording time is the time when the transaction happens in the bank's computer system.

Thinking about the bank's point of view, when the bank accepts the cheque, a payment event happens, but some time later the bank asserts the payment to the supplier and the customer on their bank statements (different assertions).

Taking the supplier's view, he accepts the cheque (payment assertion) and records the event proactively into his own computer (different recording time), although the payment event really happens several days later.

Chapter 3. Temporal Database Classifications

Most conventional databases represent the state of the dynamic real world at a single moment of time. Although the contents of the database continue to change as new information is added, these changes are viewed as modifications to the state, with the old, out-of-date data being deleted from the database. The current contents of most databases are viewed as a *snapshot* of the real world.

Temporal databases represent the progression of states of the real world over different time dimensions. In such databases, changes are viewed as additions to the information in the database. Temporal databases are thus generalizations of conventional databases and their underlying snapshot model.

Extending conventional databases into temporal databases, the models of temporal databases are different in supporting different types of time attributes (different time dimensions) and different topologies of time. In general, according to the presentation in [Snodgrass & Ahn 1985], these models can be classified into four distinct kinds of databases on the basis of supporting different time attributes. The four kinds of databases are *snapshot databases, rollback databases, historical databases,* and *temporal databases.* Unfortunately this classification means that "temporal database" is being used in two different ways: generically for all databases where time is supported, specifically for databases where event and recording time are supported.

Many non-relational database management systems are in used in commercial organisations such as banks and building societies. However, it is easier to add time to the relational model [Codd 1979] so the discussion here is limited to this model.

3.1 Snapshot Databases

In the relational model, a database is a collection of relations. Each relation consists of a set of tuples with the same set of attributes and is usually represented as a

two-dimensional table as in Figure 3.1. Only after changes in the real world being made, changes to the databases can be made. Thus a state or an instance of a database does not necessarily reflect the current status of the real world.

"Updating the state of a database is performed using data manipulation operations such as insertion, deletion, or replacement, taking effect as soon as they are committed. In this process, past states of the database, representing those of the real world object, are discarded and forgotten. We term this type of database a *snapshot database.*"[Snodgrass 1987] An entity only has one state or one observation in the snapshot database at a time. Thus, a tuple in the snapshot database refers to an entity.

Snapshot databases do not support DBMS-maintained time attributes. For example, they do not support recording time at all. However, they support some kinds of assertion times or event times in the schema for the relations. Such time attributes are not interpreted by DBMS, and their domains will appear in the schemas of relations. In relational models, snapshot databases are exactly those databases supported by the relational algebra. Unfortunately, they cannot answer queries on past states or for the future.

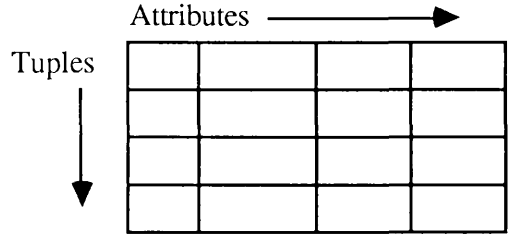


Figure 3.1 A Snapshot Relation

Taking a library model [Oxborrow 1986] as an example, we have a relation as follows,

BOOK-LOCATION (Book#, Isbn, Shelf)

The underlined attribute, Book#, is the identifier of the relation. At a certain moment an instance of the relation BOOK-LOCATION may be as shown in Figure 3.2.

Book#	Isbn	Shelf
00023	0-999-99999-0	cb001
00065	0-999-99999-0	cb003
00012	0-12345-123-x	cb002
00080	0-5656-5566-2	cb003

Figure 3.2 A BOOK-LOCATION Relation Instance

and a query in Quel [Held et al. 1975] as to the location of Book# 00023,

range of b **is** BOOK-LOCATION

retrieve (b.Shelf)

where b.Book# = "00023"

yields the result that Shelf = cb001.

This snapshot database is adequate in some applications, but inadequate in many applications. For example, it cannot answer queries such as

1) Where was Book# 00065 last July (assume that we have rearranged the books in library at least once since that time) ? (historical query)

2) How did the number of copies of each book change over the last two months? (trend analysis)

nor record facts like

3) There has been another copy Book# 00094 for the book, Isbn 0-999-99999-0, since last October. (retroactive change)

4) The book # 00080 will be moved to the shelf cb004 next month. (proactive change)

Clearly, without system support, many applications have had to maintain their temporal information in an ad-hoc manner; such operations have to be handled by specially-written application programs.

3.2 Rollback Databases

To retrieve the queries on past states, an approach can be presented in which a database is regarded as a sequence of snapshot relations indexed by recording time which serves as the time axis. Under this approach a database can be illustrated conceptually in three dimensions (see Figure 3.3 A Rollback Relation). We can view a snapshot of the database as of some moment before (*an observation*) by querying on that database along the time axis and selecting this observation. We term the operation of selecting an observation as *rollback*, and a database supporting it is termed a *rollback database*.

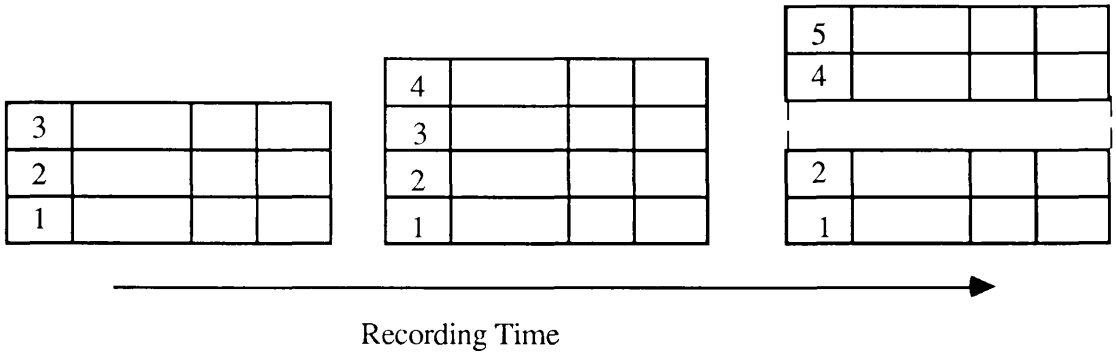


Figure 3.3 A Rollback Relation

A rollback database is a set of observations of entities in the real world. Because an entity can be observed many times, i.e., it can have many observations, one observation of an entity refers to one tuple in the rollback database. Embedding the recording time attribute, tuples in such a database are called *rollback tuples*.

Rollback databases only support recording time. By recording the history of database activity, rollback databases allow relations to be rolled back to one of their past snapshot states (observations) for querying. The distinction between snapshot databases and rollback databases is that the latter have the ability to return to any previous observation to execute a snapshot query.

One limitation of supporting recording time is that the history of database activities is recorded, rather than the history of the real world. A tuple becomes valid as

soon as it is entered into the database as in a snapshot database. Retroactive and proactive changes of events are not recorded, and errors in past observations cannot be corrected.

As an example, we extend the BOOK-LOCATION relation (a snapshot relation) into a rollback relation by embedding the recording time attribute as shown in Figure 3.4. The relation is sorted along the recording time dimension.

Recording time

(a)

Book#	Isbn	Shelf
00023	0-999-99999-0	cb001
00012	0-12345-123-x	cb002
00030	0-332-42233-1	cb004

12/4/88

(b)

Book#	Isbn	Shelf
00023	0-999-99999-0	cb001
00012	0-12345-123-x	cb002
00030	0-332-42233-1	cb004
00065	0-999-99999-0	cb001

25/6/88

(c)

Book#	Isbn	Shelf
00023	0-999-99999-0	cb001
00012	0-12345-123-x	cb002
00065	0-999-99999-0	cb003
00080	0-5656-5566-2	cb003

1/10/88

Figure 3.4 BOOK-LOCATION Relation (rollback relation version 1)

(Note: Examples are given in terms of calendar dates. In practice, the granularity of time is dependent on the application and the system. It can be other date form.)

There are three observations for the relation BOOK-LOCATION in Figure 3.4. Each presents one transaction applied to it, starting from a null relation:

(a) creation of three new entities on 12th April, 1988,

(b) addition of one entity (Book# 00065) on 25th June, 1988,

(c) deletion of one entity (Book# 00030), addition of another entity (Book# 00080) and replacement of one entity (Book# 00065 has been moved to the shelf cb003 since 15th September, 1988) on 1st October, 1988.

Note: We cannot record the fact that Book# 00065 has been moved to the shelf cb003 since 15th September, 1988 (retroactive change), because the event was recorded on 1st October, 1988.

Implementing a rollback relation in this way is impractical, due to excessive duplication: the tuples that do not change between observations must be duplicated in the new observation. Another approach that partially addresses this difficulty appends the *start* and *end* points of the recording time to each tuple, indicating the points in time when the observation of an entity was valid in the database. The relation above in this approach looks like Figure 3.5. More space is saved.

			Recording time	
Book#	Isbn	Shelf	start	end
00023	0-999-99999-0	cb001	12/4/1988	∞
00012	0-12345-123-x	cb002	12/4/1988	∞
00030	0-332-42233-1	cb004	12/4/1988	1/10/1988
00065	0-999-99999-0	cb001	25/6/1988	1/10/1988
00065	0-999-99999-0	cb003	1/10/1988	∞
00080	0-5656-5566-2	cb003	1/10/1988	∞

Figure 3.5 BOOK-LOCATION Relation (rollback relation version 2)

On the other hand, the recording (end) time attribute can be used as a deletion label to mark an observation of an entity which has been deleted or terminated. A current observation always has an infinite end time value. While a rollback tuple with a finite end recording time may represent either:

a) an observation of an entity which has been terminated in the database, such as Book# 00030;

b) an observation of an entity which has changed its state, such as Book# 00065 which was deleted from Shelf cb001 and placed on Shelf cb003.

Treating two time attributes for one time dimension is very useful in historical and temporal databases as well, because we need not really delete (move away) tuples from the database, and we can keep the observations or states of deleted entities as long as we like and view the history of the database as is/was best known to fully capture the temporal concept of TDBMS. This viewpoint will be discussed deeply in Section 3.4, Temporal databases and Chapter 4, Some Proposed Temporal Databases and Their Languages.

Finally, we can have disjoint tuples which are terminated at some time and started at another time. This could occur when a particular entity has an unknown history for a period of time.

Any query language may be converted to a query which can execute a rollback operation by adding a clause effecting the rollback. TQuel augments the retrieve statement with an **as of** clause to specify the relevant recording time (that is, to execute a rollback operation). The TQuel query

range of b **is** BOOK-LOCATION

retrieve (b.Shelf)

where b.Book# = 00065

as of "July 1988"

will find the location of Book# 00065 as it was recorded in (the beginning of) July 1988 (by the system). In this example, the result is Shelf cb001. (Note that the result of a rollback operation is a pure snapshot relation)

The concept of recording time has appeared in several systems, including TODM/TOSQL [Ariav 1986], TDB/TQuel [Snodgrass 1987], SWALLOW object store [Reed 1978], and HDB in [Clifford & Tansel 1985]. (c.f. Section 2.2.3, Figure 2.3 A Classification of Time concepts)

3.3 Historical Databases

Whereas rollback databases support recording time to represent the history of database activities, *historical databases* support assertion time, or event time, or both (logical times) to represent the history of the real world. However, assertion time and event time in historical databases are maintained by TDBMSs themselves, not maintained by users as in snapshot databases. Their domains do not appear in the schema for the relation and their semantics are provided by the database, not by the user or application program.

An historical database may also be illustrated in three dimensions (see Figure 3.6). The label of the time axis has been changed to logical time (assertion time and event time), and "the semantics are more closely related to reality, rather than update history." [Snodgrass 1987] Historical databases record a single observation per relation, storing the history as is best known. We always assume that the state of the real world being modelled does not change until next new state being recorded, thus the historical relation is an *interval* relation and the tuples in the historical databases are step-wise constants.

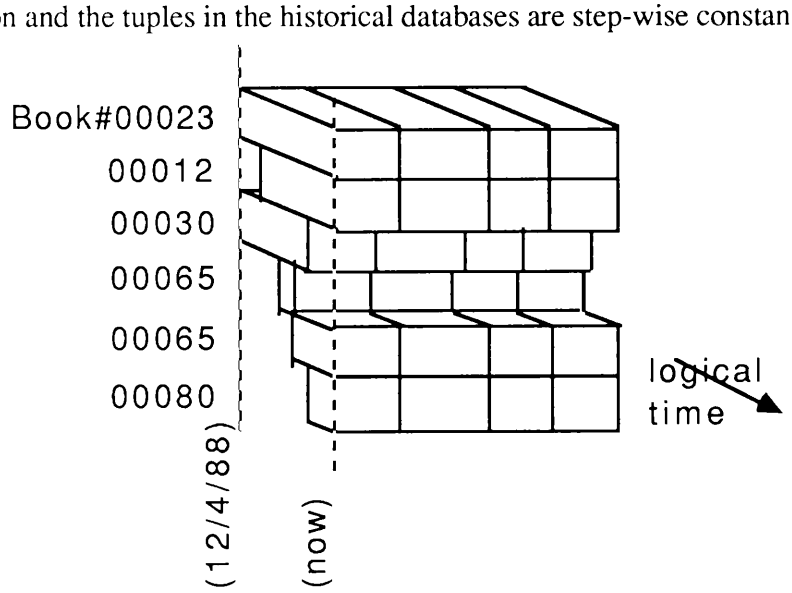


Figure 3.6 An Historical Relation

An entity can have many states along its historical dimension. Tuples in historical databases refer to *entity states* because the databases represent the history of the real world. An entity state is currently valid until the entity changes its values in the real world. When an entity changes its state, a new tuple is generated and the old one is simply terminated (but not deleted from the database). Thus, an entity can refer to many tuples, e.g., the entity Book# 00065 has two tuples in the database because it has two states in the real world.

As errors are discovered, they are corrected by modifying the database; previous states are not retained, so the database may not be viewed as it was in the past. No record is kept of the errors that have been corrected; historical databases are similar to snapshot databases in this respect.

The distinction between historical and rollback databases is that historical databases support arbitrary modification, whereas rollback databases not allow the observation of entities to be modified. Rollback databases can rollback to an incorrect previous observation; historical databases can represent current knowledge about the past. For instance, a sequence of transactions, which are almost the same as the three transactions that resulted in the rollback relation in Section 3.2 Rollback Databases, is as follows, starting from a null relation as well:

(a) addition of three new books in shelf on 12th April, 1988 (Book# 00023, Book# 00012, and Book# 00030),

(b) appendage of another new book (Book# 00065) on 25th June, 1988,

(c) termination of one book (Book# 00030), appendage of a new one (Book# 00080) and rearrangement of one book (Book# 00065 has been moved to the shelf cb003 since 15th September, 1988) on 1st October, 1988,

(d) correction of the information about Book 00012 (Book# 00012 was not available before 20th May, 1988) on 18th October, 1988.

These transactions will result in an historical relation in Figure 3.7.

Book#	Isbn	Shelf	Event time	
			from	to
00023	0-999-99999-0	cb001	12/4/1988	∞
00012	0-12345-123-x	cb002	20/5/1988	∞
00030	0-332-42233-1	cb004	12/4/1988	1/10/1988
00065	0-999-99999-0	cb001	25/6/1988	15/9/1988
00065	0-999-99999-0	cb003	15/9/1988	∞
00080	0-5656-5566-2	cb003	1/10/1988	∞

Figure 3.7 BOOK-LOCATION Relation (historical relation)

Some comments should be made that:

(1) The time axis has been changed to event time. Event time has an interval structure with the *from* time and *to* time attributes. This approach is based on the discussion about the model of rollback databases.

(2) The time attributes are now concerned with the reality of enterprises, not the reality of transactions. For instance, the updating of the entity - Book# 00030 - is expressing its termination in the real world, but not expressing the deletion from the database.

(3) There is still no way to record retroactive/proactive changes. After the transactions, the entity, Book# 00065, is valid in the shelf cb003 since 15th September, 1988, not since 1st October as in the rollback database! However, the fact that the modification was made on 1st October cannot be recorded. The reason is that there is no mechanism to record the time when an event was recorded into the database. We need two time attributes, the event time and the recording time, to fully capture the retroactive and proactive changes.

(4) There is no way to record correction activities. Book# 00012 has been changed into being valid from 20th May, 1988, but there is not any record for the previous information, Book# 00012 used to be valid from 12th April, 1988. Hence, the database was inconsistent with reality for that period of time.

(5) the entity Book# 00065 has two states in the database which are represented by the fourth and fifth tuples. Other tuples have only one state respectively in the relation at the moment.

Historical databases require more sophisticated query languages. Two such languages which have been developed are: Legol 2.0 [Jones & Mason 1980], based on the relational algebra, and TQuel [Snodgrass 1984], based on Quel [Held et al. 1975], a relational calculus query language. We will discuss them in Chapter 4. Here, we give an example, in TQuel, requesting the location of Book# 00065 on 1st October,1988, as is best known. A **when** clause is used to specify the temporal relationship of tuples participating in a derivation. The **overlap** operator specifies that the event and/or intervals overlap the time (see Chapter 4 for detail).

range of b **is** BOOK-LOCATION

retrieve (b.Shelf)

where b.Book# = 00065

when b **overlap** "1/10/1988"

The result will be Shelf cb003.

Most studies about temporal databases support historical databases as models, for example, AMPPL-II [Findler & Chen 1971], ILs [Clifford & Warren 1983], MHM [Garia & Yeung 1988], Legol 2.0 [Jones & Mason 1980] and TDB [Snodgrass & Ahn 1985]. There is a good reference for the relative researches in the paper [McKenzie 1986].

3.4 Temporal Databases

Whereas rollback databases only provide support for recording time to represent the reality of the history of database activities and historical databases support assertion time/event time to retrieve the history of the real world, we need a kind of database which can represent a relation as a sequence of the enterprise being modelled and the history of database activities. Temporal databases provide such an ability. They support all three

types of time attributes and make it possible to view tuples as being valid at some moment relative to some other moment, completely capturing the history of retroactive and proactive changes.

A temporal relation may be regarded as a sequence of historical relations indexed by recording time, each a completed historical relation indexed by logical time (assertion time/event time). Therefore a single temporal relation can be viewed as a four dimensional object as shown in Figure 3.8.

The snapshot relational database model is utilized as the underlying model of temporal database by embedding the four-dimensional temporal relation in a two-dimensional snapshot relation. There have been several ways to implement such a model proposed during the last decade. We will introduce two approaches, analysing their advantages and disadvantages, then, give some more complex examples to clarify the concepts of temporal databases well, and discuss the domain of time attributes under the approach of version 2.

The discussion will concern a temporal relation result of four transactions, the same as the sequence of events in Section 3.3, starting from a null relation:

(1) Book# 00023, Book# 00012, and Book# 00030 were stored on the shelf on 10th April, 1988 (the fact was retroactively recorded on 12th April, 1988);

(2) Book# 00065 was added to the shelf on 26th June, 1988 (the fact was proactively recorded on 25th June,1988);

(3) Book# 00030 was deleted due to being missing, and Book# 00080 was added to the shelf on 1st October; Book# 00065 has been moved from the shelf cb001 to cb003 since 15th September, 1988, and the fact was recorded on 1st October, 1988;

(4) a correction was made on 18th October, 1988, for Book# 00012 being not available before 20th May, 1988.

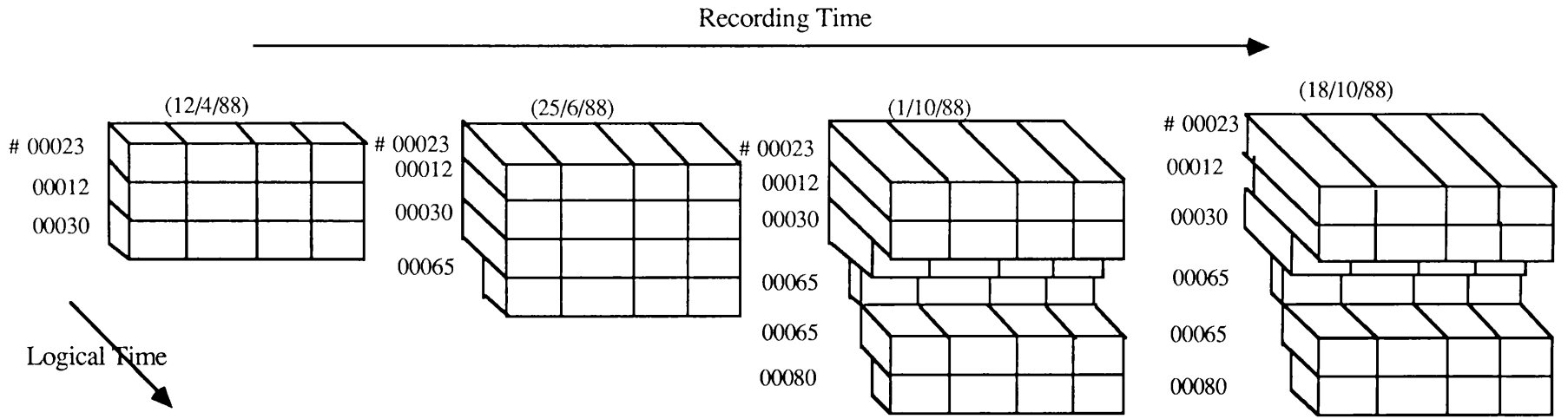


Fig. 3.8 A Temporal Relation

3.4.1 Embedding a temporal relation, version1

The most straightforward implementation is to combine two approaches shown in last two sections: appending two event time attributes to the user-defined time attributes to form a series of historical relations; each with a recording time for rollback. Figure 3.9 shows such an approach.

Recording time

Book#	Isbn	Shelf	event from time	event to time
00023	0-999-99999-0	cb001	10/4/88	∞
00012	0-12345-123-x	cb002	10/4/88	∞
00030	0-332-42233-1	cb004	10/4/88	∞

12/4/88 (a)

Book#	Isbn	Shelf	event from time	event to time
00023	0-999-99999-0	cb001	10/4/88	∞
00012	0-12345-123-x	cb002	10/4/88	∞
00030	0-332-42233-1	cb004	10/4/88	∞
00065	0-999-99999-0	cb001	26/6/88	∞

25/6/88 (b)

Book#	Isbn	Shelf	event from time	event to time
00023	0-999-99999-0	cb001	10/4/88	∞
00012	0-12345-123-x	cb002	10/4/88	∞
00030	0-332-42233-1	cb004	10/4/88	1/10/88
00065	0-999-99999-0	cb001	26/6/88	15/9/88
00065	0-999-99999-0	cb003	15/9/88	∞
00080	0-5656-5566-2	cb003	1/10/88	∞

1/10/88 (c)

Book#	Isbn	Shelf	event from time	event to time
00023	0-999-99999-0	cb001	10/4/88	∞
00012	0-12345-123-x	cb002	20/5/88	∞
00030	0-332-42233-1	cb004	10/4/88	1/10/88
00065	0-999-99999-0	cb001	26/6/88	15/9/88
00065	0-999-99999-0	cb003	15/9/88	∞
00080	0-5656-5566-2	cb003	1/10/88	∞

18/10/88 (d)

Figure 3.9 A BOOK-LOCATION Relation(temporal relation version 1)

Each update operation involves copying the previous historical relation, then applying the update to the newly created historical relation; hence, temporal relations are append only.

In such an approach, there is only one recording time attribute for each historical state. As mentioned in Section 3.2, implementing a temporal relation in this way is impractical, because the tuples that do not change between states must be duplicated in the new state. The storage requirements are increased very quickly in such an approach. However, the approach is more clear in its logical concepts than the following one which will be introduced in Section 3.4.2.

In Figure 3.9, the database has five entities in it. An entity is defined as a presentation of a real world object in the database. After being recorded into the temporal database, entities can not be removed from the database any more.

Along the recording time dimension, we can capture the observations of those entities. There are four observations for the entities: Book# 00023, Book# 0012, and Book# 00030; three observations for Book# 00065; and two observations for Book#00080. Some observations are the same because entities have not changed their values or no error has been discovered, but some are different, for instance, the observations of Book# 00012 are different in Figure 3.9 (c) and (d), because an error was discovered on 18th October: Book# 00012 was not available before 20th May.

While along the event time dimension, we can retrieve the states of some entity. For example, in Figure 3.9 (c), Book# 00065 changed its state firstly, thus, two states are presented for it. However, we should note that states of an entity do not change correspondingly with its observations. In Figure 3.9 (d), the states for Book# 00065 are the same as in (c) although there are two different observations.

To fully capture the temporal semantics, tuples in temporal databases do not correspond to the *states* or the *observations of entities*, but to the *observations of entity states*, like the fifth tuple in (c) referring to the third observation and the second state of the entity Book# 00065. There are three observations for Book# 00065 in the shelf cb001

and two for it in the shelf cb003. Thus, there are five observations of entity states for Book# 00065. We term such tuples as *temporal tuples* to mean those snapshot tuples embedded with two time dimensions: one in physical level and one in logical level and viewed in these two different ways at the same time.

Embedding temporal dimension in this way, temporal databases are not only concerned with the reality of enterprises' history, or the reality of the database transactions, but both of them. Retroactive and proactive changes can also be captured now. For instance, the event from time of the fourth tuple in Figure 3.9 (b), Book# 00065 in shelf cb001, is 26th June, 1988, while the recording time is 25th June. Therefore, the proactive change is presented; the event from time of the entity state, Book# 00023 in the shelf cb001, is 10th April, 1988, but the state was recorded two days later, i.e., 12th April, 1988. Retroactive change is captured in this way.

Meanwhile, the correction activity of Book# 00012 is captured in the database. The observation of the entity state of Book# 00012 being valid since 10th April, 1988 has become unavailable in the database since 18th October, 1988. A new tuple with the same keys is added to the relation to present a new valid interval of Book# 00012. However, the entity state, Book# 00012 is stored in the shelf cb002, is still valid in the new observation. Therefore, the event to time still has an infinite value. The correction activities of events in the database are fully recorded. We can not only retrieve the history of a particular tuple as is best known, but also can retrieve it as was known at some other moment.

3.4.2 Embedding a temporal relation, version2

Another approach introduced in [Snodgrass 1987] to embed the temporal dimension in a snapshot relation is to append four time attributes, two each denoting intervals of event and recording time. The BOOK-LOCATION temporal relation is illustrated in Figure 3.10.

Book#	Isbn	Shelf	Event time		Recording time	
			from	to	Start	end
00023	0-999-99999-0	cb001	10/4/88	∞	12/4/88	∞
00012	0-12345-123-x	cb002	10/4/88	∞	12/4/88	18/10/88
00012	0-12345-123-x	cb002	20/5/88	∞	18/10/88	∞
00030	0-332-42233-1	cb004	10/4/88	∞	12/4/88	1/10/88
00030	0-332-42233-1	cb004	10/4/88	1/10/88	1/10/88	∞
00065	0-999-99999-0	cb001	26/6/88	∞	25/6/88	1/10/88
00065	0-999-99999-0	cb001	26/6/88	15/9/88	1/10/88	∞
00065	0-999-99999-0	cb003	15/9/88	∞	1/10/88	∞
00080	0-5656-5566-2	cb003	1/10/88	∞	1/10/88	∞

Figure 3.10 A BOOK-LOCATION Relation (temporal relation version 2)

In the version 2, the semantics of entity, entity state, observation of an entity and observation of an entity state are not as visual as in the version 1. Some tuples only present the observations of the related entities individually, like the second and the third tuples. They are different observations, but present the same state of Book# 00012; some tuples not only present the observations of the relative entities but also those entity states, like the tuples of book# 00065; while some tuples present not only both observations and states of entities but the entities themselves, for instance, the first and the last tuples. However, it is clear that each tuple still presents one observation of an entity state of the related entity.

Looking at the tuples of Book# 00065, the sixth and the seventh tuples present the first state of the entity Book# 00065, and the eighth tuple presents the second state; while in the system view, the sixth tuple presents the first observation of this entity, but the seventh and the eighth tuples represent the second one. Thus, some comments can be made that:

• In the same *observation*, the tuples relating to an entity have the same recording start time and at most have one infinite *event to time*;

• In the same *state*, the values of the same user-defined attributes are the same and only one infinite *recording end time* exists for this entity state.

In such an approach, tuples are assumed to be coalesced in that tuples with identical values for the explicit attributes neither overlap nor are adjacent in time. Specially, the tuples of an entity should be continuous in recording time, the system time.

Examples: a sequence of examples is treated here to explain the concepts of the temporal database model (version 2) and how the modification operations can be carried out on such an model. The discussion continues from the temporal relation in Figure 3.10 and focuses on the evolution in the entity of Book# 00030. Thus, a relation drawn from Figure 3.10 is

			Event time		Recording time	
Book#	Isbn	Shelf	from	to	Start	end
00030	0-332-42233-1	cb004	10/4/88	∞	12/4/88	1/10/88
00030	0-332-42233-1	cb004	10/4/88	1/10/88	1/10/88	∞

1) If on the 18th October, 1988, Book# 00030 was found actually being unavailable on 10th September, then the relation is modified to

			Event time		Recording time	
Book#	Isbn	Shelf	from	to	Start	end
00030	0-332-42233-1	cb004	10/4/88	∞	12/4/88	1/10/88
00030	0-332-42233-1	cb004	10/4/88	1/10/88	1/10/88	18/10/88
00030	0-332-42233-1	cb004	10/4/88	10/9/88	18/10/88	∞

2) Assume that Book# 00030 was found again on 25th October and put back into the shelf and database at the same date. A new tuple should be appended to the database to state the new entity state of Book# 00030 in this case, because the early state is still available in the database. Thus, the third tuple keeps an infinite value for the recording end time.

Book#	Isbn	Shelf	Event time		Recording time	
			from	to	Start	end
00030	0-332-42233-1	cb004	10/4/88	∞	12/4/88	1/10/88
00030	0-332-42233-1	cb004	10/4/88	1/10/88	1/10/88	18/10/88
00030	0-332-42233-1	cb004	10/4/88	10/9/88	18/10/88	∞
00030	0-332-42233-1	cb004	25/10/88	∞	25/10/88	∞

Now we have two infinite recording end times for the same entity. However, two different states of this entity have been explained. Retrieving the information about the Book# 00030 as best known, two states can be obtained, one for the early life of that book, and one for the later life.

3) On 12th November, we discovered that Book# 00030 was really lost on 15th September. We make a correction to the third tuple, and the relation will look like

Book#	Isbn	Shelf	Event time		Recording time	
			from	to	Start	end
00030	0-332-42233-1	cb004	10/4/88	∞	12/4/88	1/10/88
00030	0-332-42233-1	cb004	10/4/88	1/10/88	1/10/88	18/10/88
00030	0-332-42233-1	cb004	10/4/88	10/9/88	18/10/88	12/11/88
00030	0-332-42233-1	cb004	10/4/88	15/9/88	12/11/88	∞
00030	0-332-42233-1	cb004	25/10/88	∞	25/10/88	∞

4) Suppose Book# 00030 was really found on 20th October, and the fact was discovered on 15th November, then, the relation is modified into

Book#	Isbn	Shelf	Event time		Recording time	
			from	to	Start	end
00030	0-332-42233-1	cb004	10/4/88	∞	12/4/88	1/10/88
00030	0-332-42233-1	cb004	10/4/88	1/10/88	1/10/88	18/10/88
00030	0-332-42233-1	cb004	10/4/88	10/9/88	18/10/88	12/11/88
00030	0-332-42233-1	cb004	10/4/88	15/9/88	12/11/88	∞
00030	0-332-42233-1	cb004	25/10/88	∞	25/10/88	15/11/88
00030	0-332-42233-1	cb004	20/10/88	∞	15/11/88	∞

5) Finally, we discovered on 20th November that Book# 00030 was never lost, then we must make a modification to the relation as follows

			Event time		Recording time	
Book#	Isbn	Shelf	from	to	Start	end
00030	0-332-42233-1	cb004	10/4/88	∞	12/4/88	1/10/88
00030	0-332-42233-1	cb004	10/4/88	1/10/88	1/10/88	18/10/88
00030	0-332-42233-1	cb004	10/4/88	10/9/88	18/10/88	12/11/88
00030	0-332-42233-1	cb004	10/4/88	15/9/88	12/11/88	20/11/88
00030	0-332-42233-1	cb004	25/10/88	∞	25/10/88	15/11/88
00030	0-332-42233-1	cb004	20/10/88	∞	15/11/88	20/11/88
00030	0-332-42233-1	cb004	10/4/88	∞	20/11/88	∞

There is only one entity state in 5), although there were two entity states in 4)! This is because the relation only represents one state during the lifespan of the entity Book#00030 finally. However, there are seven observations for this entity.

The evolution history of the book 00030 was fully captured in this series of examples. The examples also give out all possible values for the time attributes. We will discuss the domain of time attributes in next section.

3.4.3 The domain for time attributes

There are two time attributes, each containing two time values, in the temporal database model of version 2. They are

event from time: denoting the beginning of an entity state in one observation;

event to time: denoting the end of an entity state in one observation;

recording start time: the beginning of an observation of an entity state in the database; and

recording end time: the end of an observation of an entity state in the database.

The beginning of the times, both event time and recording time, always has a finite value, because temporal databases actually are still concerned with historical (past or present) data. Historical data have finite values for their beginning of time intervals. (If

future data were to be considered, the domain for the beginning of time attributes should include an infinite (or unknown) value to identify an object being not available at the moment till sometime in the future.) Both event to time and recording end time can have a finite value or an infinite value.

Any reasonable time value can be represented as a finite integer value. A finite time value (f) cannot be changed at all after being recorded into the database. However, an infinite time value will tend to be changed. Only considering historical data, four possible combinations of time values can be obtain as shown in Figure 3.11.

	from	to	start	end
1)	f_f	∞	f_s	∞
2)	f_f	∞	f_s	f_e
3)	f_f	f_t	f_s	∞
4)	f_f	f_t	f_s	f_e

Figure 3.11 The Combinations of Time Values

1) An entity has been valid in the real world since the time f_f, and has existed in the database since the time f_s.

2) An entity was thought currently valid by the system during the interval f_s to f_e. That is, the observation for a current valid entity state was only valid from f_s to f_e.

3) An entity has changed its state in the real world, but the observation for this state is still valid in the database.

4) An entity has changed its state in the real world and the relative observation is considered available in the database only until the time f_e.

The values of event from time and recording start time denote the retroactive or proactive change of an event in a database, while the values of event to time and recording end time denote whether an entity state or an observation is still available in the real world or in the database. For example, if

$f_f = f_s$, then the entity state was recorded when it started;

$f_f > f_s$, then the entity state was proactively recorded into the database;

$f_f < f_s$, then the entity state was retroactively recorded in the database;

$f_t = \infty$, the state is still active;

$f_c = \infty$, the observation is still available;

f_t = finite integer, the state has been changed;

f_c = finite integer, the observation has been superseded.

Therefore we can fully capture the whole history of retroactive/proactive changes into temporal databases, and retrieve the temporal information of an object.

3.4.4 Entity, entity state, observation of entity, observation of entity state and tuples

A representation of *a distinguishable object* in the real world is widely termed as an *entity* in databases; while the term *tuple* in a database corresponds approximately to the notion of a *flat record instance*. That is, tuple is a physical concept and entity is a logical one.

An entity can have many states along the event time dimension and can be observed many times along the recording time dimension. Along both time dimensions at the same time, we view an entity through the observations of its entity states.

In the snapshot database, no time dimension exists. Only one instance of an entity is captured. A tuple can refer to an observation of this entity, also to a state of this entity, and therefore to the entity itself as well. An update to the snapshot database will cause a change to the entity state and an observation of that entity at the same time.

In the historical database, a tuple refers to an entity state because the database may present many states for one entity. While in the rollback database, a tuple refers to an observation of the entity to capture different observations of the entity along the recording time dimension. Of course, updates cause changes to entity states in historical databases and to observations of entities in rollback databases.

In the temporal databases, an entity is viewed in both time dimensions. To fully present the temporal activity of the entity, a tuple should refer to an observation of one entity state of the entity. However, a tuple in any type of database always represents one single physical record.

To sum up the discussion, a simple diagram shown in Figure 3.12 presents the relationships between entity, entity state, observation of entity, observation of entity state and tuple in four different databases. t denotes a single tuple, T a set of tuples, Ts a subset of tuples concerned with an entity state, and To a subset of tuples concerned with an observation of an entity; E means entity, Es means entity state, Eo means observation of entity, and Eos means observation of entity state. X denotes an impossible expression.

	snapshot	rollback	historical	temporal
E	t	T	T	T
Es	t	X	t	Ts
Eo	t	t	X	To
Eos	t	X	X	t

Figure 3.12 The Relationships between Entity, Entity State, Observation of Entity, Observation of Entity State and Tuples

3.4.5 Merging temporal tuples

In order to reduce the amount of space required for storing temporal information it is possible to consider merging tuples. We will approach this by means of an example.

After the transaction on 1st October, Book# 00065 has three tuples in the database, as shown below,

			Event time		Recording time	
Book#	Isbn	Shelf	from	to	Start	end
00065	0-999-99999-0	cb001	26/6/88	∞	25/6/88	1/10/88
00065	0-999-99999-0	cb001	26/6/88	15/9/88	1/10/88	∞
00065	0-999-99999-0	cb003	15/9/88	∞	1/10/88	∞

The first and the second tuples present the same first entity state, Book# 00065 in Shelf cb001. These two tuples can be combined together without changing any semantics for them by the following rule:

if ordinary time attributes of tuples, which are related to the same entity, have values in the form of Figure 3.13

from	to	start	end
a	∞	b	c
a	d	c	∞
d	∞	c	∞

Figure 3.13 Time Attributes of Tuples to be Merged

(This means that in the first state, the first tuple is followed continuously by another tuple which is obtained in the same transaction of recording the next state for this event.)

then the intervals of first two tuples can be merged together to produce Figure 3.14:

from	to	start	end
a	d	b	∞
d	∞	c	∞

Figure 3.14 Merged Tuples' Time Attributes

In the example of Book# 00065, the following result is obtained when the tuples are merged.

			Event time		Recording time	
Book#	Isbn	Shelf	from	to	Start	end
00065	0-999-99999-0	cb001	26/6/88	15/9/88	25/6/88	∞
00065	0-999-99999-0	cb003	15/9/88	∞	1/10/88	∞

We can know that, from the table above, the book 00065 has a period living in Shelf cb001 and since 15th September, it has begun a new lifespan in Shelf cb003. The fact that the termination of the early life of Book# 00065 was into database on 1st October can be retrieved by the recording start time of the last tuple. Under such an approach, more storage can be saved.

However, merging cannot be applied everywhere. For instance, the temporal tuples for Book# 00030 in the original example of Figure 3.10 are

			Event time		Recording time	
Book#	Isbn	Shelf	from	to	Start	end
00030	0-332-42233-1	cb004	10/4/88	∞	12/4/88	1/10/88
00030	0-332-42233-1	cb004	10/4/88	1/10/88	1/10/88	∞

If the two tuples are merged, the result will be

			Event time		Recording time	
Book#	Isbn	Shelf	from	to	Start	end
00030	0-332-42233-1	cb004	10/4/88	1/10/88	12/4/88	∞

We have lost the information about when the termination was recorded into the database totally, because there is no tuple following for the later life of Book# 00030. Suppose another transaction specifying the later life of Book# 00030 is made again: Book# 00030 was found in the same date, 1st October, but the fact was recorded on 25th October, then the lifespan of Book# 00030 was recorded as

			Event time		Recording time	
Book#	Isbn	Shelf	from	to	Start	end
00030	0-332-42233-1	cb004	10/4/88	∞	12/4/88	1/10/88
00030	0-332-42233-1	cb004	10/4/88	1/10/88	1/10/88	∞
00030	0-332-42233-1	cb004	1/10/88	∞	25/10/88	∞

We still cannot merge the early two tuples, because the last two tuples are different observations of the entity.

Therefore, the condition for merging tuples is two tuples which are relative to the same entity state are followed by another tuple which is the same observation as the second tuple (i.e., the third tuple has a recording start time: c), and presents the new state of the same entity (i.e., the third tuple must be coalesced with the second tuple in event time and have an event from time: d).

3.5 Some Temporal Queries

Based on the four kinds of distinct relations given above, we represent some queries in TQuel to highlight the differences and similarities among the four types of relations, snapshot, rollback, historical, and temporal. We use the qualifiers: snapshot, rollback, historical and temporal to distinguish different relations.

(1) Where is the book# 00065?

To answer this question, the query for the snapshot relation should be as

```
range of b is BOOK-LOCATION.snapshot
retrieve (b.Shelf)
where b.Book# = "00065"
```

the query for the rollback relation should be

```
range of b is BOOK-LOCATION.rollback
retrieve (b.Shelf)
where b.Book# = "00065"
as of "now"
```

and the historical query is

```
range of b is BOOK-LOCATION.historical
retrieve (b.Shelf)
where b.Book# = "00065"
when b overlap "now"
```

while the query for the temporal relation (temporal query) is

range of b **is** BOOK-LOCATION.temporal

retrieve (b.Shelf)

where b.Book# = "00065"

when b **overlap** "now"

as of "now",

which manipulates both kinds of time in a query.

Four queries have the same result, Shelf cb003, but in rather different ways. For the snapshot relation, the query searches the whole relation for the derived tuples; for the rollback relation, the current snapshot state is used; for the historical relation, the tuples currently active are searched; and for the temporal relation, the tuples currently active in the current historical state are searched.

(2) Where was the book# 00065 as known by the system on 20th October?

This is a query to execute a rollback operation. It cannot be applied to either the snapshot relation or the historical relation, because they do not support the system time attribute, recording time. The query for the rollback relation will be

range of b **is** BOOK-LOCATION.rollback

retrieve (b.Shelf)

where b.Book# = "00065"

as of "20/10/88".

Thus, the result will be Shelf cb003.

However, for the temporal relation, the query is

range of b **is** BOOK-LOCATION.temporal

retrieve (b.Shelf)

where b.Book# = "00065"

when b **overlap** b

as of "20/10/88"

The result is Shelf cb003 or cb001! The derived relation is a historical relation as shown in Figure 3.15.

	Event time	
Shelf	from	to
cb001	26/6/88	15/9/88
cb003	15/9/88	∞

Figure 3.15 A Derived Historical Relation -- BOOK65

Assume that the query was carried out on 25th November, 1988 (one recording time). We must note that such a relation is only valid for the best known by the system on 20th October, 1988 (a second recording time); and two entity states were recorded into the system on 1st October, 1988 (a third recording time). There is no mechanism to record such system times. Thus the derived view cannot be taken as a temporal relation to carry further temporal queries[1]. More semantics study is needed to present a mechanism to record them. We will discuss a little bit about it in Chapter 6, Conclusion and Further Research.

(3) Where was the book# 00065 on 20th October?

For this query, we cannot get any result from either the snapshot or rollback relations, because neither of them support the event time at all. However, we can have queries as follows to apply to the historical relation and temporal relation.

For the historical relation, the query is

range of b **is** BOOK-LOCATION.historical

retrieve (b.Shelf)

where b.Book# = "00065"

when b **overlap** "20/10/88"

[1] In [Snodgrass & Ahn 1985], the authors stated that the result is a temporal relation, and in [Snodgrass 1987], a semantics was supported in retrieve tuple calculus statement to derive a temporal relation. However, the semantics in the two articles are different. The former presented a mechanism to record the third recording time only, while the latter supported a semantics to record the first recording time. None supports the second one or all of them. To support three kinds of recording time together, the schema evolution should be discussed.

The result should be Shelf cb003 for the historical query.

For the temporal query

range of b **is** BOOK-LOCATION.temporal

retrieve (b.Shelf)

where b.Book# = "00065"

when b **overlap** "20/10/88"

as of "now"

the result will be Shelf cb003 too.

(4) Where was the book# 00065 on 20th September as was known by the system on 20th October?

Only a temporal relation can answer such a query, because it supports both rollback operation and historical query; the rollback operation on a temporal relation selects a particular historical state, on which a historical query may be performed. The query is

range of b **is** BOOK-LOCATION.temporal

retrieve (b.Shelf)

where b.Book# = "00065"

when b **overlap** "20/9/88"

as of "20/10/88"

The result is Shelf cb003. If we issue the query as

range of b **is** BOOK-LOCATION.temporal

retrieve (b.Shelf)

where b.Book# = "00065"

when b **overlap** "20/9/88"

as of "20/9/88"

that is, to retrieve the information about the location of Book# 00065 on 20th September as was known by the system at the same date, the result, Shelf cb001, is different from the former.

There have been some models around this approach. The well-known one is TDB presented by Snodgrass et al. [Snodgrass & Ahn 1985]. Its query language is a temporal language which supports both logical time and recording time. The model also supports all four kinds of databases. TRM (Time Relational Model) is another example of a temporal database [Ben-Zvi 1982]. However, the query language defined for TRM is not a temporal query language, because it can derive only snapshot relations. In the model AIM, Lum et al. propose two time concepts, logical time and physical time, but the model is limited to only the design and implementation aspect of the physical time. MHM (MultiHomogeneous Model) [Gadia & Yeung 1988] proposes a generalized relation model for a temporal database which allows time stamping with respect to a Boolean algebra of multidimensional time stamps. As an application, a two dimensional model which allows objects with real world and transaction oriented time stamps can be used to query the past states of databases. It can also be used to give a precise classification of the errors and updates in a database, and is a promising approach for querying these errors and updates.

Chapter 4 Some Proposed Temporal Databases and Their Languages

Temporal databases have been developed using many database models. A list of ongoing projects and bibliography can be found in [Snodgrass & McKenzie 1986] and [Snodgrass 1986]. In these two papers, at least 25 research groups studying time in databases were mentioned. The research activity may be classified loosely into three emphases: the formulation of a semantics of time at the conceptual level, the development of a model for TDBMS, analogous to the relational model for snapshot databases, and the design of temporal query languages.

The model of TDB proposed by Snodgrass et al. was presented in a series of papers [Snodgrass 1984, Snodgrass & Ahn 1985, Snodgrass & Ahn 1986, Mckenzie 1986, Snodgrass 1986, Mckenzie & Snodgrass 1987A, Mckenzie & Snodgrass 1987B, Mckenzie & Snodgrass 1987C, Snodgrass & Gomez 1986]. A temporal query language TQuel, a superset of Quel, was defined and formalized in the papers [Snodgrass 1984, Snodgrass 1987]. A prototype of the temporal database management system was built by extending Ingres to support the temporal query language TQuel [Ahn 1986, Ahn & Snodgrass 1986].

In this chapter, we shall roughly introduce the model of TDB and note its differences with the proposed general temporal databases models in Chapter 3, although we use the same classification for the temporal databases. We pay special attention to TDB's query language - TQuel and will introduce three temporal clauses and temporal predicate operators and constructors which play important roles in TQuel.

The introductions in Section 4.1 and Section 4.2 will be taken as a background material for the discussion in Chapter 5. In Section 4.3, two other temporal database models will be introduced as well. The differences among three languages will be

compared in Section 4.4 to emphasize the need of taking TDB(TQuel) as an underlying model for the discussion about temporal databases.

4.1 The model of TDB

There are at least two possible approaches to the development of a relational model for TDBMS which have been suggested. One is to extend the semantics of the relational model to incorporate time directly. The other is to base a TDBMS on the snapshot model, with time appearing as additional attributes. The second approach was taken in the papers above in modelling TDBs: utilizing the snapshot model. The *snapshot* relational database model is used as the underlying model of TDB by embedding the four-dimensional temporal relation in a two-dimensional snapshot relation. This approach has also been introduced in Chapter 3 to express the basic concepts of four kinds of temporal databases. The difference between them is that the TDB in the University of North Carolina is composed of a sequence of observations indexed by *transaction time*, with each observation consisting of *a sequence of snapshot slices* indexed by *valid time*, while the model in Section 3.4 is composed of a sequence of observations indexed by *recording time*, with each observation being *an interval relation* indexed by *logical time* (or event time).

4.2 The Query Language of TDB - TQuel

4.2.1. The basic model - Quel

The language of TDB, TQuel, is designed to be a minimal extension, both syntactically and semantically, of Quel. All legal Quel statements are also valid TQuel statements, and such statements have identical constructs defined in Quel and TQuel when the time domain is fixed, and the additional constructs defined in TQuel to handle time have direct analogues in Quel. The three additional temporal constructs (**valid, when** and **as**

of clauses) defined in TQuel were shown to be direct semantic analogues of Quel's **where** clause and target list.

4.2.1.1 Quel retrieve statement

Quel is the query language of a relational data base and graphics system, INGRES (INteractive Graphics and Retrieval System) [Held et al. 1975]. It is a calculus based language. Each query of Quel contains one or more Range-Statements and one or more Retrieve-Statements. The retrieve statement consists of two basic components: the *target list*, specifying how the attributes of the relation being derived are computed from the attributes of the underlying relations, and a *where clause*, specifying which tuples participate in the derivation. The form of a query in Quel can be outlined as

Query

= {Range-Statement}{Retrieve-Statement}

Range-Statement

= **range of** {Variable} **is** Relation

Retrieve-Statement

= **retrieve** [**into** Result-name] (Target-List)

where Qualification

Target-List

= {Result-Domain = Function}

Qualification

= expression of attribute-references and literals through comparison operators (e.g., =, >, <), or Boolean operators (AND, OR, and NOT).

Note: { } denotes "one or more", and [ ] denotes "zero or one".

The goal of a query is to create a new relation for each Retrieve-Statement. The relation so created is named by the "Result-Name" clause or by default a temporary relation which is displayed (Result-Name is optional), and the domains in that relation are named by the "Result-Domain" names given in the Target-List. To create the desired relation, first

consider the product of the ranges of all variables which appear in the Target-List and the Qualification of the Retrieve-Statement. Each term in the Target-List is a function and the Qualification is a truth function, i.e., a function with values true or false, on the product space. The desired relation is created by evaluating the Target-List on the subset of the product space for which the Qualification is true, and eliminating duplicate tuples.

Example 4.1.1

Relation: BOOK-LOAN (Book#, Isbn, Name, Address, Date-due-back)

Query: Find the book numbers of all books which have been borrowed by Peter and should be returned in five days.

range of b **is** BOOK-LOAN

retrieve into Book-Return (Book# = b.Book#)

where b.Name = "Peter"

and b.Date-due-back <= "5/12/88"

Note: Assume the transaction took place on 1/12/88.

4.2.1.2 Tuple relational calculus statements for Quel

Let $D_1, D_2, ..., D_n$ be nonempty sets, not necessarily distinct. A subset R of the product $D_1 \times D_2 \times ... D_n$ is called a relation, and D_i are called the domains of R. Let u be an element of R, then u is an n-tuple $(u_1, u_2, ..., u_n)$ where u_i belongs to D_i. Then, tuple relational calculus statements are of the form

$$\{t^{(i)} \mid \psi(t)\}$$

where the variable t denotes a tuple of arity i, and $\psi(t)$ is a first-order predicate calculus expression containing only the free tuple variable t. Then $\psi(t)$ defines the tuple u contained in the relation R specified by the Quel statement. The tuple calculus statement for the skeletal Quel retrieve statement

range of t_1 **is** R_1

. . . .

range of t_k **is** R_k

retrieve $(t_{i_1}.D_{j_1}, \ldots, t_{i_r}.D_{j_r})$

where ψ

is

$$\{u^{(r)} \mid (\exists t_1) \ldots (\exists t_k)(R_1(t_1) \wedge \ldots \wedge R_k(t_k)$$
$$\wedge u[1] = t_{i_1}[j_1] \wedge \ldots \wedge u[r] = t_{i_r}[j_r]$$
$$\wedge \psi')\}$$

which states that each t_i is in R_i, that each result tuple u is composed of r particular components, that the m th attribute of u is equal to the j_mth attribute (having an attribute name of D_{jm}) of the tuple variable t_{jm}, and that the condition ψ' (ψ modified for attribute names and Quel syntax conventions) holds for u. The first line corresponds to the relevant range statements, the second to the target list, and the third to the **where** clause.

4.2.2 Temporal clauses, predicate operators and constructors in TQuel

Based on Quel, TQuel extends Quel in time dimension by adding three additional temporal constructs, **valid**, **when**, and **as of** clauses.

4.2.2.1 Three temporal clauses of TQuel statements[1]

Valid, when and **as of** clauses are three additional temporal constructs defined in TQuel. They operate on two time attributes, *valid time* and *transaction time*.

1) The **when** clause

"The **when** clause is the temporal analogue to Quel's **where** clause."[Snodgrass 1987] A **when** predicate specifies the temporal relationship of tuples participating in a derivation (i.e., retrieves the valid tuples on implicit valid time attributes). This clause consists of the keyword **when** followed by a *temporal predicate* on the tuple variable

[1] TQuel statements are presented in Appendix C, and the syntax of TQuel is stated in Appendix A. Appendix B presents the defaults of the three temporal clauses.

representing the implicit time attributes of the associated relations. The syntax is similar to *path expressions*, which are regular expressions augmented with parallel operators.

Taking the snapshot relation BOOK-LOAN in Example 4.1.1 as a basic model, we embed two time dimensions, valid time and transaction time, into it and obtain a temporal relation which contains the data about who borrowed books as shown in Figure 4.1.

				Date-due	Valid Time		Trans. Time	
Book#	Isbn	Name	Address	-back	from	to	start	stop
00023	0-999-99999-0	Peter	Park Avn.	12/6/88	12/4/88	∞	12/4/88	23/5/88
00023	0-999-99999-0	Peter	Park Avn.	12/6/88	12/4/88	23/5/88	23/5/88	∞
00023	0-999-99999-0	Tony	Kelvin St.	12/12/88	6/10/88	∞	6/10/88	∞
00012	0-12345-123-x	Peter	Park Avn.	3/12/88	9/9/88	∞	10/9/88	∞
00030	0-332-42233-1	Rob	Kelvin St.	10/11/88	10/9/88	∞	10/9/88	1/10/88
00030	0-332-42233-1	Rob	Kelvin St.	10/11/88	10/9/88	1/10/88	1/10/88	∞
00065	0-999-99999-0	Peter	Park Avn.	9/12/88	5/10/88	∞	6/10/88	∞
00080	0-5656-5566-2	Tony	Kelvin St.	12/12/88	6/10/88	∞	6/10/88	7/11/88
00080	0-5656-5566-2	Tony	Kelvin St.	12/12/88	6/10/88	4/11/88	7/11/88	∞

Figure 4.1 A Temporal Relation

A query listing the books borrowed by Peter in November 1988 from Figure 4.1 is shown as follows, which employs the **when** clause to specify an **overlap** relationship of tuples. The result is Book# 00012 and 00065.

Example 4.2.1 **range of** b **is** BOOK-LOAN

retrieve (Book# = b.Book#)

where b.Name = "Peter"

when b **overlap** "11/88"

The query was carried on the current historical relation by the default. The **overlap** operator specifies that tuples' valid time intervals overlap the time we are concerned with (November 1988). It will be defined in Section 4.2.2.2. Another example of the **when** clause follows:

Example 4.2.2 What books were borrowed by Peter this year (1988)?

range of b **is** BOOK-LOAN

retrieve into BookbyPeter(Book# = b.Book#, Isbn = b.Isbn)

where b.Name = "Peter"

when "1/1/88" **precede** (**begin of** b)

and (**end of** b) **precede** "now"

precede is another temporal predicate operator, and **begin of** and **end of** are temporal unary constructors. All of them will be discussed in Section 4.2.2.2 as well. While **and** is a logical operator as normal. The result of this query is Book# 00023, 00012 and 00065. Although the book 00023 has been returned by Peter (the borrowing activity has been terminated), it can also be picked up by this query. It is impossible for snapshot databases to pick up such a tuple. It must be very clear that the query was carried out on the current historical relation as well.

2) The **valid** clause

"The **valid** clause serves the same purpose as the target list: specifying the value of an attribute in the derived relation"[Snodgrass 1987] and how the time during which the derived tuples are valid is computed. "If the derived relation is to be an event relation, the **valid at** variant specifies the value of the single time in the temporal attribute." "The variant **valid from . . . to . . .** is used when the derived relation is to be an interval relation."[Snodgrass 1987]

TQuel supports historical queries by augmenting the retrieve statement with a **valid** clause and a **when** predicate. The difference between **when** and **valid** clauses is the **when** clause retrieves the implicit valid time attributes for a derivation, while the **valid** clause states the new valid time in a derivation.

The following query employs a **valid** clause to specify the new valid time of derived relation.

Example 4.2.3 Modify the BOOK-LOAN relation (Peter renewed the book 00012 on 1st December 1988).

range of b **is** BOOK-LOAN

replace b (Date-due-back = "1/1/89")

valid from begin of "12/88"

where b.Book# = "00012" **and** b.Name = "Peter"

The result is shown in Figure 4.2.

Book#	Isbn	Name	Address	Date-due -back	Valid Time from	Valid Time to	Trans. Time start	Trans. Time stop	
00023	0-999-99999-0	Peter	Park Avn.	12/6/88	12/4/88	∞	12/4/88	23/5/88	
00023	0-999-99999-0	Peter	Park Avn.	12/6/88	12/4/88	23/5/88	23/5/88	∞	
00023	0-999-99999-0	Tony	Kelvin St.	12/12/88	6/10/88	∞	6/10/88	∞	
00030	0-332-42233-1	Rob	Kelvin St.	10/11/88	10/9/88	∞	10/9/88	1/10/88	
00030	0-332-42233-1	Rob	Kelvin St.	10/11/88	10/9/88	1/10/88	1/10/88	∞	
00065	0-999-99999-0	Peter	Park Avn.	9/12/88	5/10/88	∞	6/10/88	∞	
00080	0-5656-5566-2	Tony	Kelvin St.	12/12/88	6/10/88	∞	6/10/88	7/11/88	
00080	0-5656-5566-2	Tony	Kelvin St.	12/12/88	6/10/88	4/11/88	7/11/88	∞	
00012	0-12345-123-x	Peter	Park Avn.	3/12/88	9/9/88	∞	10/9/88	1/12/88	*
00012	0-12345-123-x	Peter	Park Avn.	3/12/88	9/9/88	1/12/88	1/12/88	∞	*
00012	0-12345-123-x	Peter	Park Avn.	1/1/89	1/12/88	∞	1/12/88	∞	*

Figure 4.2 A Derived Relation

Note: * marks the modified tuples.

3) The **as of** clause

"The **as of** clause is similar to the **where** and **when** clauses, in that it provides an additional constraint on the underlying tuples participating in the query."[Snodgrass 1987] It specifies the relevant transaction time and supports the rollback operation. The rollback operation on a temporal relation selects a particular historical state, on which a historical query can be executed.

Example 4.2.4 Who borrowed the book 00030 as best known in September 1988?

range of b **is** BOOK-LOAN

retrieve (Name = b.Name)

where b.Book# = "00030"

as of begin of "9/88" **through end of** "9/88"

The result is Rob, although such a tuple has been terminated in the relation.

Valid and **when** clauses can be employed in TQuel modification statements, whereas **as of** clause only supports rollback retrieve. For modification statements, the **as of** clause is fixed as the default **as of** "now", representing the most recent state.

The relationships among eight distinct retrieve queries and four kinds of databases can be presented in Figure 4.3 to explain the functions of three temporal constructs. Assume that the derived times are intervals.

Derived Query relations / Databases	with: valid	with: when	with: as-of	with: valid when	with: valid as-of	with: when as-of	with: valid when as-of	with: none of them
Snapshot	×	×	×	×	×	×	×	snap.
Rollback	×	×	snap.	×	×	×	×	snap.
Historical	hist.	hist.	×	hist.	×	×	×	snap.
Temporal	hist.	hist.	hist.	hist.	hist.	hist.	hist.	hist.

Figure 4.3 Temporal Queries and Temporal Databases

The column headings show which temporal clauses (**valid**, **when**, or **as of**) appear in the query. The queries without a visible **as of** clause actually take a default clause **as of** "now" (c.f. Appendix C). The row labels state four distinct databases which are to be operated on. The symbol × means an impossible operation.

If a TQuel statement does not contain a **valid**, **when** and **as of** clause, then it looks identical to the analogous standard Quel retrieve statement; thus it should have an identical semantics, and can be applied to any database. The derived relation can be considered as a snapshot relation (except to temporal databases) as best known now.

Queries with **valid** or **when** clauses cannot operate on snapshot or rollback databases, but can be applied to historical databases and the derived relation is a historical relation. So, further historical relations can be derived again from it. Queries only with **as**

of clause cannot be applied to snapshot or historical databases, but can be applied to rollback databases. The derived relation will be a snapshot relation as best known of some moment in time. Any kind of retrieve queries applied to temporal databases will derive an historical relation as a result which can only be further used to answer historical queries. It could be presented as a temporal relation, but to present recording time for it, more semantics discussion are needed (c.f., Chapter 6 The Conclusion). However, if the queries state modification operation, then the result will become a temporal relation according to the semantics for modification queries (c.f., Chapter 5 or Appendix C)

4.2.2.2 Temporal predicate operators and temporal constructors

The three temporal clauses employ four temporal constructors and three predicate operators in temporal expressions to specify the derived time attributes. The temporal expressions are E-expressions and I-expressions. We will discuss them in turn.

1) E-expressions and I-expressions

TQuel supports both event time and interval time. E-expressions and I-expressions are employed to deal with these two times.

"An *e-expression* is simply an expression containing tuple variables, temporal constants, and temporal constructors, with the constraint that the expression must result in an event. E-expressions are used in the **valid** and **as of** clauses. Since the **as of** clause specifies rollback to a particular transaction time, the e-expression in an **as of** clause must evaluate to a temporal constant. An equivalent constraint is that an e-expression within an **as of** clause must not contain a tuple variable."[Snodgrass 1987]

"An *i-expression* is an expression containing tuple variables, temporal constants, and temporal constructors that evaluates to an interval."[Snodgrass 1987] An i-expression can employ e-expressions, and an e-expression can also employ i-expressions. In the following **valid** clause, we give a simple example of e-expressions and i-expressions. The **valid** clause

valid at begin of (f1 **overlap** a)

specifies that the time value returned should be the first instant when both tuples are valid. In this clause, f1 **overlap** a is an i-expression which returns an interval time, while **begin of** (f1 **overlap** a) is an e-expression which returns a time point. E-expressions must have **begin of** or **end of** constructors as top-level operators.

2) Temporal constructors

"A *temporal constructor* is a unary or binary operator that takes one or two events or intervals as arguments and returns an event or interval." "The unary prefix temporal constructors are **begin of** and **end of**, both returning events. The binary infix temporal constructors are **overlap** and **extend**, both returning intervals."[Snodgarss 1987]

Assume the Before predicate as: Before(α, β) ::= $\alpha \leq \beta$, then the temporal constructors were defined as follows after defining a few auxiliary functions on integers (*First, Last*) and tuple variables (*event, interval*) in [Snodgrass 1987]:

First (α, β) = α (if *Before* (α, β)), or β otherwise

Last (α, β) = β (if *Before* (α, β)), or α otherwise

event (t) = $<t_{at}, t_{at}>$

interval (t) = $<t_{from}, t_{to}>$

beginof ($<\alpha, \beta>$) = $<\alpha, \alpha>$

endof ($<\alpha, \beta>$)=$<\beta, \beta>$

overlap ($<\alpha, \beta>$,$<\gamma, \delta>$)=$<$*Last* (α, γ), *First* (β, δ)$>$

extend ($<\alpha, \beta>$,$<\gamma, \delta>$)=$<$*First* (α, γ), *Last* (β, δ)$>$

3) Temporal predicate operators

"A *temporal predicate operator* is a binary infix operator that takes events or intervals as arguments and returns a Boolean value. The three temporal predicate operators are **precede, overlap,** and **equal**."[Snodgrass 1987]

The temporal predicate operators are replaced by the analogous predicate Before on ordered pairs of integers as follows:

precede ($<\alpha, \beta >$,$<\gamma, \delta >$) = *Before* (β, γ)

overlap ($<\alpha, \beta >$,$<\gamma, \delta >$) = *Before* (α, δ)$\wedge$*Before* (γ, β)

equal (<α, β >,<γ, δ >) = *Before* (α, γ)∧*Before* (γ, α)

∧*Before* (β, δ)∧*Before* (δ, β)

The logical operators (**and, or, not**) are allowed as well in temporal expressions except the e-expression and i-expression.

4) Temporal predicate

A *temporal predicate* is an expression containing logical operators (**and, or, not**) operating on expressions containing temporal predicate operators (**precede, overlap,** or **equal**), operating on e-expressions and i-expressions. Temporal predicates are used only in **when** clauses. The temporal predicate in the **when** clause determines whether the tuples may participate in the derivation by examining their relative order. This predicate is generated in three steps: First, the tuple variables and the temporal constructors are replaced by the functions defined in the previous subsection. Second, the **and, or,** and **not** operators are replaced by the logical predicates. Finally, the temporal predicate operators are replaced by analogous predicates on ordered pairs of integers.

As an example, applying the first step to the temporal predicate

(**begin of** (a **overlap** b) **precede** (**end of** (b **extend** c))) **or** (c **precede** a)

where, a ::= <α, β>, b ::= <γ, δ>, and c ::= <θ, λ>

results in

→ (**beginof**(**overlap**(<α, β>, <γ, δ>)) **precede**(**extend**(<γ, δ>, <θ, λ>)))

or (<θ, λ> **precede** <α, β>)

→ (**beginof** (<Last (α, γ), First (β, δ)>) **precede**

(**endof** (<First (γ, θ), Last (δ, λ)>))) **or** (<θ, λ> **precede** <α, β>)

→ (<Last (α, γ), Last (α, γ)> **precede** (<Last (δ, λ), Last (δ, λ)>))

or (<θ, λ> **precede** <α, β>).

The second step results in

(<Last (α, γ), Last (α, γ)> **precede** (<Last (δ, λ), Last (δ, λ)>))

∨ (<θ, λ> **precede** <α, β>),

and third step results in

Before (Last (α, γ), Last (δ, λ)) $\vee$ Before (λ, α).

4.3 Two Other Temporal Models

TODM [Ariav 1986] and Legol [Jones & Mason 1980] are two other temporal relational database management systems similar to TDB. They share the same philosophy on modelling: embedding the time dimension into traditional relational databases (snapshot databases) as time sequence attributes. However, there are some differences between them in semantics and syntax. In this section, we are going to discuss their similarities and differences with TDB. TODM was chosen for comparison because it supports an extension of SQL, the "standard" relational query language. Legol 2.0 is included because it is widely referenced as an early source of ideas on temporal database representation.

In the approach of embedding additional temporal attributes into snapshot databases, the logic of the model does not incorporate time at all; instead, the query language must translate queries and updates involving time into retrievals and modifications on the underlying snapshot relations. In particular, the query language must provide the appropriate values for these attributes in the relation being derived. Therefore, the discussion will be concerned with the query languages of these three database systems.

4.3.1 The model of TODM

TODM is considered as a cube, which has two standard relational dimensions as well: *object* and *attribute*, and a dense and continuous representation of the temporal dimension of the relation. The time dimension reflects the system time, *recording time* (RT). The state of the relation at any point in time is determined by slicing the cube horizontally in a depth corresponding to the specified time, and observing the values of data that prevail for each of the objects represented in the relation. The following figures explain the same database, BOOK-LOCATION from Figure 3.4, in TODM form. The

database is recorded as a series of snapshot relations sequentially along the RT time dimension in Figure 4.4. (Note: Figure 4.4 (a) - (c) explain a whole database).

Book#	Isbn	Shelf
00023	0-999-99999-0	cb001
00012	0-12345-123-x	cb002
00030	0-332-42233-1	cb004

(a) The relation on 12/4/88

Book#	Isbn	Shelf
00023	0-999-99999-0	cb001
00012	0-12345-123-x	cb002
00030	0-332-42233-1	cb004
00065	0-999-99999-0	cb001

(b) The relation on 25/6/88

Book#	Isbn	Shelf
00023	0-999-99999-0	cb001
00012	0-12345-123-x	cb002
00065	0-999-99999-0	cb003
00080	0-5656-5566-2	cb003

(c) The relation on 1/10/88

Figure 4.4 The BOOK-LOCATION Relation in TODM Form

Such a database is called a rollback database rather than a temporal database, because the event time dimension has not been presented here. Thus, the model is considered as a three time dimensional object. To deal with temporal databases, event time is taken as an user-defined time in TODM. For example, the database, BOOK-LOCATION, in TDB temporal version is presented as Figure 4.5. And it can be presented as a series of historical relations in TODM form in Figure 4.6 (a) - (c):

Book#	Isbn	Shelf	Event time		Recording time	
			from	to	Start	end
00023	0-999-99999-0	cb001	10/4/88	∞	12/4/88	∞
00012	0-12345-123-x	cb002	10/4/88	∞	12/4/88	∞
00030	0-332-42233-1	cb004	10/4/88	∞	12/4/88	1/10/88
00030	0-332-42233-1	cb004	10/4/88	1/10/88	1/10/88	∞
00065	0-999-99999-0	cb001	26/6/88	∞	25/6/88	1/10/88
00065	0-999-99999-0	cb001	26/6/88	15/9/88	1/10/88	∞
00065	0-999-99999-0	cb003	15/9/88	∞	1/10/88	∞
00080	0-5656-5566-2	cb003	1/10/88	∞	1/10/88	∞

Figure 4.5 A Temporal Database (BOOK-LOCATION)

Book#	Isbn	Shelf	event time
00023	0-999-99999-0	cb001	10/4/88
00012	0-12345-123-x	cb002	10/4/88
00030	0-332-42233-1	cb004	10/4/88

(a) The relation on 12/4/88

Book#	Isbn	Shelf	event time
00023	0-999-99999-0	cb001	10/4/88
00012	0-12345-123-x	cb002	10/4/88
00030	0-332-42233-1	cb004	10/4/88
00065	0-999-99999-0	cb001	26/6/88

(b) The relation on 25/6/88

Book#	Isbn	Shelf	event time
00023	0-999-99999-0	cb001	10/4/88
00012	0-12345-123-x	cb002	10/4/88
00065	0-999-99999-0	cb003	15/9/88
00080	0-5656-5566-2	cb003	1/10/88

(c) The relation on 1/10/88

Figure 4.6 BOOK-LOCATION Temporal Database in TODM Form

Dealing with event time in this way, TODM can be considered as a four-dimensional object as well.

Comparing Figure 4.6 with Figure 4.5, it can be seen that TODM concerns itself with event relations. It contributes only one attribute for each time dimension. However,

the temporal database in Figure 4.5 is a series of interval relations. As stated in Chapter 3 (The Temporal Database Classifications), in interval relations, supporting only one attribute for one time dimension is impractical due to the information duplication.

In more detail, we also can find that some information was lost in the TODM approach. First, there is no mechanism to record when Book# 00030 terminated its lifespan in the real world. The point to note is that the event time and the recording time can be different.

Second, an event which has been terminated but its information is still available in the database cannot be explained properly (e.g. Book# 00065 in Shelf cb001). Now no change can be made to the event, Book# 00065 used to be in Shelf cb001. If such information were kept in the database, we cannot explain when does this event terminate in the real world. The fact is that we only have one event time attribute which states the event start time. Therefore, only those events which have been valid and are still valid (active) can be recorded. Such an approach can satisfy the application of presenting the history of active events, but cannot satisfy the need to capture the activity of dead events in databases.

Third, considering the recording time, the termination of a tuple can not be properly explained as well. For instance, if we found the book 00030 again on 18th November and recorded it into the database two days after, then the database in the Figure 4.6 should be extended by another relation as shown in Figure 4.7.

Book#	Isbn	Shelf	event time
00023	0-999-99999-0	cb001	10/4/88
00012	0-12345-123-x	cb002	10/4//88
00065	0-999-99999-0	cb003	15/9/88
00080	0-5656-5566-2	cb003	1/10/88
00030	0-332-42233-1	cb004	18/11/88

The relation on 20/11/88

Figure 4.7 The Relation on 20/11/88

Now we will make confusion when we make the change for Book# 00030 in the database. Book# 00030 appeared in Figure 4.6 (a) and (b), disappeared from (c) and then reappeared as shown in Figure 4.7. This causes confusion because it is not obvious what happened to Book# 00030 between 25/6/88 and 20/11/88. We cannot properly identify the lifespans of the Book# 00030 in the database. Clearly, to represent interval relations, one attribute for one time dimension is not enough.

Fortunately, TODM can be expanded to process *multiple* user-defined time dimensions (c.f., Section 4.4 Language comparisons). More than one user-defined time attribute can be handled within TODM. Taking *event to time* as another user-defined time, the problem of capturing the event termination information as discussed above can be solved. On the other hand, if we consider the termination of transaction presenting another *entity state*, the recording (*end of*) time can be considered as the recording *start* time of new entity state. Therefore, TODM can handle interval relations properly as well.

For instance, the temporal database in Figure 4.5 will be explained as shown in Figure 4.8 in TODM form:

Recording time 12/4/88

Book#	Isbn	Shelf	event from time	event to time
00023	0-999-99999-0	cb001	10/4/88	∞
00012	0-12345-123-x	cb002	10/4/88	∞
00030	0-332-42233-1	cb004	10/4/88	∞

25/6/88

Book#	Isbn	Shelf	event from time	event to time
00023	0-999-99999-0	cb001	10/4/88	∞
00012	0-12345-123-x	cb002	10/4/88	∞
00030	0-332-42233-1	cb004	10/4/88	∞
00065	0-999-99999-0	cb001	26/6/88	∞

1/10/88

Book#	Isbn	Shelf	event from time	event to time
00023	0-999-99999-0	cb001	10/4/88	∞
00012	0-12345-123-x	cb002	10/4/88	∞
00030	0-332-42233-1	cb004	10/4/88	1/10/88
00030	0-332-42233-1	--	1/10/88	∞
00065	0-999-99999-0	cb001	26/6/88	15/9/88
00065	0-999-99999-0	cb003	15/9/88	∞
00080	0-5656-5566-2	cb003	1/10/88	∞

Figure 4.8 The Temporal Database in TODM (BOOK-LOCATION)

This almost has the same form as the temporal databases in Figure 3.9. Of course, excessive duplications have not been reduced to minimum. Meanwhile, due to determining the time duration over which an entity state is valid, successive observations have to be examined. This will slow down the performance of TODM and create complications with operations [Snodgrass 1987].

4.3.2 The model of Legol

A database in Legol is defined as a pure three dimensional object: a traditional relational flat file (or table) embedded with a time dimension. Along this time dimension, the history of events can be captured. Legol has not provided another time dimension (recording time) to capture the evolution of the database itself. Therefore, Legol cannot be considered as temporal-complete. A database in Legol can only be considered as a historical database, rather than a temporal database.

By extension to the basic relational idea, the time dimension has two attributes: *start* and *end* dates, defining the whole period of existence associated with the information in each table entry. The end time can be undefined to identify the current information with the historical information which has a defined end time value. For example, a historical database, BOOK-LOCATION, in Figure 3.7 can be explained in Legol as Figure 4.9.

Book-Location	(Book#	Isbn)	start	end
cb001	00012	0-999-99999-0	12/4/1988	--
cb002	00023	0-12345-123-x	20/5/1988	--
cb004	00030	0-332-42233-1	12/4/1988	1/10/1988
cb001	00065	0-999-99999-0	25/6/1988	15/9/1988
cb003	00065	0-999-99999-0	15/9/1988	--
cb003	00080	0-5656-5566-2	1/10/1988	--

Figure 4.9 The BOOK-LOCATION Database in Legol

The leftmost attribute is called the "characteristic" in Legol; it has a label which is the same as the name given to the relation and may be used by default in operations where

only a single attribute is required. In Figure 3.7, it is the attribute "Shelf". The other two attributes are called "identifiers" in Legol. A reference to the whole relation would take the form Book-Location (Book# Isbn) including the time attributes by default.

A point to note is that although Legol can deal with other user-defined times as well, it has not presented any mechanism to operate on these time attributes at the system level. Thus, Legol cannot be considered as a multi-dimensional object.

4.4 Language Comparisons

TDB supports a temporal query language: TQuel, which is based on the model of the Quel language; TODM supports a SQL-like language, TOSQL; while Legol 2.0 is a temporal query language for Legol.

To compare TQuel with other relational temporal query languages, Snodgrass suggested 17 characteristics to check their properties. Four of the characteristics are essential according to Snodgrass.

4.4.1 Four basic properties

4.4.1.1 A formal semantics

The first requirement is that a temporal query language must be well defined. It should have a formal semantics. Without a formal semantics, the meaning of each construct, and the interaction between constructs, is unclear.

TQuel is formalized using the tuple calculus [Ullman 1982]. It has a formal semantics for the retrieve statement because embedding temporal dimensions into retrieve tuple calculus causes no problems. However, there were some semantic problems with the modification statements which will be discussed in Chapter 5. We will improve the semantics of modification statements in TQuel by introducing a new definition for *Before* predicate and using thirteen temporal relationships defined by Allen. Thus, TQuel can be said to have a formal semantics.

Unfortunately, TOSQL and Legol 2.0 have not presented formal semantics.

4.4.1.2 Supporting historical queries

TQuel, TOSQL and Legol 2.0 also support historical queries. Historical queries are the queries which can be formulated that derive information valid at a point in time from information in underlying relations valid at other points in time. Two aspects are captured under such a definition: the ability to refer to the time that the information was valid and the ability to perform "join-like" operations on logical time over multiple relations.

For the first aspect, all three languages automatically meet the needs because all of them can handle logical time:

1) TQuel accomplishes this through its **valid** and **when** clauses;

Example 4.4.1 in TQuel: What books have been borrowed by Peter this year?

We employ a temporal database below for the query:

BOOK-LOAN (Book#, Isbn, Name):

Book#	Isbn	Name	from	to	start	end

The query is:

range of b **is** BOOK-LOAN

retrieve into BookbyPeter(Book# = b.Book#, Isbn = b.Isbn)

where b.Name = "Peter"

when b **overlap** "1988"

Assume the year is 1988, and the transaction was executed on 15th November.

2) TOSQL supports a special temporal component *<time-spec>* to deal with user-defined time inclusive event time through AT, WHILE, DURING, BEFORE and AFTER operators (see Appendix D Syntax specifications of TOSQL for detail);

Example 4.4.1 in TOSQL:

The model will be changed into:

BOOK-LOAN {Book#, Isbn, Name, event-from-time, event-to-time, RT}

and the query is:

SELECT b.Book#, b.Isbn

INTO BookbyPeter.Book#, BookbyPeter.Isbn

FROM BOOK-LOAN b

WHERE b.Name = "Peter"

WHILE event-from-time ≤ 15/11/88 AND event-to-time ≥ 1/1/88

DURING (-∞ -- +∞) ALONG event-from-time, event-to-time

AS-OF PRESENT ALONG RT

Note:*event-from-time* and *event-to-time* define the valid time interval of event and are assumed to be logical time dimensions for the relation BOOK-LOAN. RT stands for the recording time (i.e., transaction time in TQuel) dimension.

3) Legol 2.0 supports historical queries via the *while*, *since*, *until*, and *during* operators;

Example 4.4.1 in Legol 2.0:

The model for BOOK-LOAN database in Legol form is:

BOOK-LOAN (Isbn Name)	start end

while the query becomes a succinct rule as below:

BookbyPeter (Isbn) <= Book-Loan (Isbn, "Peter")

while YEAR

Note: "<=" is an update symbol. BookbyPeter not only is a new view, but also presents the characteristic of the view. It is actually the attribute, Book#, in the TQuel model. The "while" is a time intersect operator. This can be considered as a relational join which also identifies overlapping time periods. YEAR is a single entry table defining the derived time interval for the query:

YEAR	start	end
1988	1/1/88	15/11/88

For the second aspect, TQuel via Quel-like tuple calculus processes multiple relations on valid time because the tuple calculus in Quel can deal with multiple relations (c.f., Section 4.2.1.2).

TOSQL can be satisfied with this property[1] as well. In a query (Appendix D. Syntax specifications of TOSQL), the component <time-spec> specifies the derived logical time intervals:

<query> ::= <b-query><obj-spec><time-spec><time-qualif>

<time-spec> ::= <time-period><time-dimension>

<time-dimension> ::= ALONG RT | ALONG <tsa>

<tsa> means Time-Stamp Attribute which standards for both logical time and physical time dimensions. "In general, a relation scheme has at least one TSA, the internally controlled RT (Recording-Time, i.e., physical time), but it no doubt may have more than one TSA." [Ariav 1986] This means that <tsa> can be explained as:

<tsa1, tsa2, ..., tsan>, or

<tsa list>

to satisfy with the need of operating on multiple time dimensions.

Considering that SQL can process multiple relations via the explicit range variable names [Date 1987] and TOSQL is a minimal extension of SQL, TOSQL should be able to process multiple relations as well.

Example 4.4.2. What books have been borrowed by staff during April 1988? Suppose we have two relative relations:

BOOK-LOAN: Book#, Isbn, Name ; and

EMPLOYMENT: Employment, Name.

1) In TQuel, the model will be as following temporal relations:

[1] In [Snodgrass 1987], Snodgrass criticized: TOSQL falls short because only one relation may participate in the query, although aggregates, which are only mentioned, may provide a measure of valid-time support.

BOOK-LOAN (Book#, Isbn, Name):

Book#	Isbn	Name	from	to	start	end

EMPLOYMENT (Employment, Name):

Employment	Name	from	to	start	end

and the query is:

range of a **is** BOOK-LOAN

range of b **is** EMPLOYMENT

retrieve into BookbyStaff (Book# = a.Book#, Isbn = a.Isbn)

where a.Name = b.Name **and** b.Employment = "staff"

when (a **overlap** b) **and** (a **overlap** "4/88")

as of "now"

2) In TOSQL, the model and the query will be taken as the following forms:

The model,

BOOK-LOAN {Book#, Isbn, Name, event-from-time, event-to-time, RT}

EMPLOYMENT {Employment, Name, event-from-time, event-to-time, RT};

The query,

SELECT a.Book#, a.Isbn

INTO BookbyStaff.Book#, BookbyStaff.Isbn

FROM BOOK-LOAN a, EMPLOYMENT b

WHERE a.Name = b.Name, AND b.Employment = "staff"

WHILE a.event-from-time ≤ b.event-to-time

AND a.event-to-time ≥ b.event-from-time

AND a.event-from-time ≤ 30/4/88

AND a.event-to-time ≥ 1/4/88

DURING (-∞ -- +∞) ALONG a.event-from-time,

a.event-to-time, b.event-from-time, b.event-to-time

AS-OF PRESENT ALONG RT

3) As stated above, the "while" operator supplies a temporal join operation to enable Legol 2.0 to process multiple relations. The model and query for Example 4.4.2 in Legol 2.0 are:

BOOK-LOAN (Isbn	Name)	start	end

EMPLOYMENT (Name)	start	end

BookbyStaff (Isbn) <= BOOK-LOAN (Isbn, Name)

while EMPLOYMENT (Name) = "staff" while (MONTH)

Note: MONTH is a table defining the valid time interval.

MONTH	start	end
April	1/4/88	30/4/88

4.4.1.3 Rollback transaction

A temporal query language must support rollback, and hence physical time. A query language supporting historical queries but not rollback is properly termed historical, rather than temporal. Only TQuel and TOSQL support rollback, both through **as-of** clauses. Legol 2.0 does not support physical time, thus, it is a historical query language.

An example for TQuel is as follows:

Example 4.4.3 Who borrowed the book 00030 as best known September 1988?

range of b **is** BOOK-LOAN

retrieve into Book#30(Name = b.Name)

where b.Book# = "00030"

as of "9/88"

While TOSQL supports the following query for the same example:

```
SELECT b.Name
INTO Book#30.Name
FROM BOOK-LOAN b
WHERE b.Book# = "00030"
AS-OF "9/88" ALONG RT
```

4.4.1.4 Implementable

TDB and Legol both have a prototype implementation ([Ahn & Snodgrass 1986] and [Jones & Mason 1980]). No prototype for TODM has been presented in the published literature, but some research has been carried out [Shiftan 1986].

4.4.2 The where, while, and when clauses

TQuel modifies the retrieve statement of Quel to include two additional components that deal with the temporal aspects of the query, namely, (1) a definition of the "mechanism" by which the implicit time attributes of the derived relation are to be constructed from the corresponding attributes of the source relations, i.e., the **valid** clause, and (2) a temporal conditional: a temporal predicate concerning the implicit time attributes of the associated relation, i.e., the **when** clause.

The **when** clause is the temporal analogue to Quel's **where** clause. **Where** clause selects the derived objects, while **when** clause singles periods of time in which the derived objects existed. The interaction between this "**when**" clause and the regular "**where**" in a query is through a logical AND relationship.

TOSQL employs WHERE and WHILE clauses as the basis for selecting source tuples. WHERE plays almost the same role as the **where** clause in TQuel to qualify the derived tuples, while WHILE clause, similarly with the **when** clause in TQuel, identifies the valid interval in which the derived tuples existed. However, they are different to each other in some sense.

They are different in specification syntax.

The **where** clause in TQuel is a general first-order predicate calculus expression in Quel to define the derived tuples. It consists of only attribute-references and literals through comparison operators or Boolean operators. The **when** clause consists of the keyword, **when**, followed by a *temporal* predicate on the tuple variable of representing the implicit *time attributes* of the associated relations;

But, in TOSQL, WHERE and WHILE clauses include a similar type of expression:

WHERE <selection-expression><prevalence-mode>

WHILE <selection-expression><temp-boundaries>

<selection-expression> relates attribute-references and literals through

comparison operators or Boolean operators

<prevalence-mode> ::= EVERYWHEN | SOMEWHEN

<temp-boundaries> ::= DURING (-∞ -- +∞) | DURING (<t> -- <t>)

Of cause, WHERE and WHILE clauses are different as stated before. WHERE is embedded in the temporal context eventually designated by WHILE.

On the other hand, TQuel and TOSQL are different in handling multiple relations. Time attributes are explicit (visible) in WHERE and WHILE clauses and handled in a nearly ad-hoc manner. One example shows that the **where** and **when** clauses in TQuel seem more sensible than the WHERE and WHILE in TOSQL. The example is translating a query, which is drawn out from [Snodgrass 1987], from TQuel form to TOSQL form. The query is based on two temporal databases:

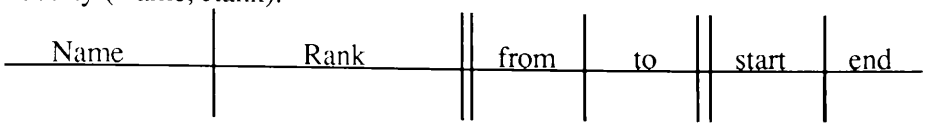
Faculty: Name, Rank ; and

Associates: Name

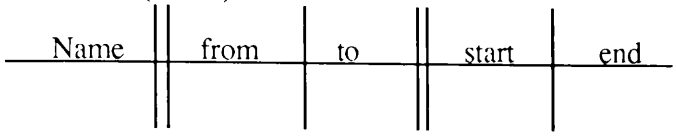
Example 4.4.4 Who got promoted from assistant to full professor while at least one other faculty member remained at the associate rank ?

The model of databases in TDB takes the form as below:

Faculty (Name, Rank):



Associates (Name):



and the query in TQuel should be:

range of f1 **is** Faculty

range of f2 **is** Faculty

range of a **is** Associates

retrieve into stars (Name = f1.Name)

valid from (**begin of** f1) **to** (**end of** f2)

where f1.Name = f2.Name **and** f1.Rank = "Assistant"

and f2.Rank = "Full"

when *(f1* ***overlap*** *a)* ***and*** *(f2* ***overlap*** *a)*

as of "now"

The model for the databases in TDBM form is:

Faculty {Name, Rank, from-time, to-time, RT}; and

Associates {Name, from-time, to-time, RT}.

The query in TOSQL is:

SELECT F1.Name

INTO stars.Name

FROM Faculty F1, Faculty F2, Associates

WHERE F1.Rank="Assistant" AND F2.Rank = "Full"

AND F1.Name = F2.Name

AND NOT(F1.Name = Associates.Name)

WHILE F1.from-time < Associates.to-time

AND F1.to-time ≥ Associates.from-time

AND F2.from-time < Associates.to-time

AND F2.to-time ≥ Associates.from-time

DURING (-∞ -- +∞) ALONG F1.from-time, F1.to-time,

F2.from-time, F2.to-time,

Associates.from-time, Associates.to-time

AS-OF PRESENT ALONG F1.RT, F2.RT, Associates.RT

F1.RT, F2.RT and Associates.RT are recording time attributes for each relation. Other time attributes are taken as user-defined times. Clearly, in TOSQL, time attributes are still maintained in a nearly ad-hoc manner.

In TQuel, we note that neither is there any statement in the **where** clause to link the relation Associates with the other two relations nor is there in the **when** clause. However, in TOSQL query, due to taking event time as user-defined time, more specifications are needed for maintaining TOSQL statements. We must find out a primary key in Associates relation and a foreign key in F1 relation to link three relations. If not, we cannot identify the relative tuples in Associates relation. In this special example, the primary key of Associates relation is the attribute "Name". Correspondingly, the foreign key of F1 is the attribute "Name" as well. Thus, the clause, *AND NOT(F1.Name = Associates.Name)*, is used in the WHERE clause, but this is not sufficient in the case where *Associates* is empty. It should be noted that there is not any general rule for finding out a primary key. But, the **overlap** operators in the **when** clause of TQuel can fetch relative tuples among multiple relations automatically! Thus, the **when** clause seems more intelligent than is immediately obvious.

Legol 2.0 selects source tuples mainly via the *while* operator. The *while* operator not only supports an intersection over derived tuples, but also identifies all overlapping periods of time on relative time attributes. It combines two functions, object selection and temporal selection, together. Actually, it plays a role of *temporal join* on relative relations. Take Example 4.4.2 as an example, and suppose three basic relations have values as the following tables:

BOOK-LOAN (Isbn		Name)	start	end
00023	0-999-99999-0	Peter	12/4/88	23/5/88
00012	0-12345-123-x	Peter	9/9/88	--
00030	0-332-42233-1	Rob	10/9/88	1/10/88
00065	0-999-99999-0	Peter	5/10/88	--
00080	0-5656-5566-2	Tony	6/10/88	9/11/88
00023	0-999-99999-0	Tony	6/10/88	--

EMPLOYMENT (Name)		start	end
student	Rob	25/9/86	10/6/88
staff	Tony	11/2/87	--
staff	Peter	1/9/87	--
staff	Rob	10/6/88	--

MONTH	start	end
April	1/4/88	30/4/88

The query is:

BookbyStaff (Isbn) <= BOOK-LOAN (Isbn, Name)

while EMPLOYMENT (Name) = "staff" while (MONTH)

Here, the equality operator " = " selects the derived tuples from the EMPLOYMENT (Name) relation:

staff	Tony	11/2/87	--
staff	Peter	1/9/87	--
staff	Rob	10/6/88	--

while "while (MONTH)" intersects out two new derived tuples and identifies valid time intervals from the derived relation:

staff	Tony	1/4/88	30/4/88
staff	Peter	1/4/88	30/4/88

The valid durations for the two derived tuples are set as the overlapping intervals between two relations.

The first "while" operator in the statement,

BOOK-LOAN (Isbn, Name) while EMPLOYMENT (Name) = "staff"

while (MONTH),

joins the relation BOOK-LOAN and the derived relation above and identifies all overlapping periods of time. The derived tuple is

00023 0-999-99999-0 Peter staff || 12/4/88 30/4/88

Update operator "<=" projects two specified attributes into the final derived relation, BookbyStaff. The result will be:

00023 0-999-99999-0 || 12/4/88 30/4/88

4.4.3 Dealing with disjoint time intervals

Sometime, dealing with disjoint intervals is necessary. For example, to answer the query: what books have been borrowed during the months of March in each of the years 1985-1989? the retrieve statement should be able to select out a series of disjoint intervals which covers the range of time from the beginning of the earliest interval to the end of the latest one.

Legol 2.0 can deal with such queries, because the derived intervals are specified as a table. Each disjoint interval is stated as an entity in the table. And the query can process each entity as one of derived time intervals. To answer the query above, a rule in Legol 2.0 is:

Book (Isbn) <= BOOK-LOAN (Isbn, Name) while (MONTH)

The table of MONTH is:

MONTH	start	end
March85	1/3/85	31/3/85
March86	1/3/86	31/3/86
March87	1/3/87	31/3/87
March88	1/3/88	31/3/88
March89	1/3/89	31/3/89

TOSQL is said to be able to deal with both continuous time and disjoint intervals [Ariav 1986 p:509]. But actually it has not presented a specification syntax to deal with

disjoint intervals. It only supports the query which processes event time (time point), or coalesced intervals. For instance, ideally we would like to write the query as follows:

```
SELECT a.Book#, a.Isbn
INTO Book.Book#, Book.Isbn
FROM BOOK-LOAN a
WHILE a.event-from-time ≥ "1/3/*" AND a.event-to-time ≤ "31/3/*"
DURING (1985 -- 1989) ALONG a.event-from-time, a.event-to-time
AS-OF PRESENT ALONG RT
```

However, the time specifications "1/3/*" and "31/3/*" do not appear to be legal. We have to deal with disjoint intervals in an ad-hoc manner. The query above has to be explained as:

```
SELECT a.Book#, a.Isbn
INTO Book.Book#, Book.Isbn
FROM BOOK-LOAN a
WHILE (a.event-from-time ≥ "1/3/85"
        AND a.event-to-time < "31/3/85")
        OR (a.event-from-time ≥ "1/3/86"
        AND a.event-to-time < "31/3/86")
        OR (a.event-from-time ≥ "1/3/87"
        AND a.event-to-time < "31/3/87")
        OR (a.event-from-time ≥ "1/3/88"
        AND a.event-to-time < "31/3/88")
        OR (a.event-from-time ≥ "1/3/89"
        AND a.event-to-time < "31/3/89")
DURING (-∞ -- +∞) ALONG a.event-from-time, a.event-to-time
AS-OF PRESENT ALONG RT
```

Tuples are assumed to be coalesced in TQuel. Such tuples with identical values for the explicit attributes neither overlap nor are adjacent in time. To keep this attribute with derived relations, TQuel has not supported operations on disjoint intervals. To deal with disjoint intervals, TQuel has to maintain its query in an ad-hoc manner as in TOSQL.

range of a **is** BOOK-LOAN

retrieve into Book (Book# = a.Book#, Isbn = a.Isbn)

when (**begin of** a ≥ "1/3/85" **and end of** a < "31/3/85")

or (**begin of** a ≥ "1/3/86" **and end of** a < "31/3/86")

or (**begin of** a ≥ "1/3/87" **and end of** a < "31/3/87")

or (**begin of** a ≥ "1/3/88" **and end of** a < "31/3/88")

or (**begin of** a ≥ "1/3/89" **and end of** a < "31/3/89")

as of "now"

4.4.4 Summary

Comparing TQuel with other two temporal languages on four main criteria and the mechanisms handling time attributes, TQuel shows itself more interesting and sensible than the other two.

It has a well defined formal semantics. The temporal concepts were developed on both retrieval semantics and operational semantics. Under such definitions, the temporal meaning of each construct in the language and the interaction between constructs are clear.

TDB supports real temporal retrieve operations to interval temporal databases, both in logical time dimension and physical time dimension. In both dimensions, time is maintained by DBMS itself (i.e., at the system level). The values of temporal domains are not handled by users, but interpreted by the system automatically.

Time attributes are implicit in the language. The implicit time attributes of the derived relation can be constructed directly from the corresponding attributes of the source relations because temporal predicates concern the implicit time attributes of the associated relations directly.

Finally, TQuel can handle multiple relations more properly. It is not necessary to include the primary key and foreign key in an ad-hoc manner. More criteria and discussions of comparisons can be found in the paper [Snodgrass 1987].

Chapter 5 New Semantics for TQuel's Modification Statements

A temporal query language must be well defined. More specifically, it should have a formal retrieval semantics and a well-defined operational semantics. Without a formal semantics, the meaning of each construct, and the interaction between constructs, is unclear. TQuel is formalized using the tuple calculus with temporal predicates to capture the temporal semantics of query. The extension is natural on the retrieve statement but it is not well-defined on the modification statements. The problems first come out from the basic temporal predicate *Before*.

In this chapter, we are going to argue that the basic predicate *Before* causes an indeterminacy problem, in Section 5.1, and introduce the relationships between Allen's time interval relations and TQuel's temporal predicate operators, in Section 5.2, as an outline of new semantics of TQuel modification statements to avoid semantic confusions. In the following section, Section 5.3, the modification statements will be discussed and the semantics of TQuel modification statement will be developed and redefined. In particular, event modification statements are going to be presented. We will present the new definitions for the *Before* predicate and the new semantics for the modification statements at the same time.

5.1 The *Before* Predicate in TQuel

5.1.1 The problem of the Before predicate

The *Before* predicate was defined as a linear order "≤" predicate on integer time values, i.e., Before = $\{t_i, t_j \mid t_i$ is less than or equal to $t_j\}$. It means that one event is *before* a second event if the time value of the first, expressed as an integer or real value, is less than or equal to (≤) the time value of the second. This set served as the underlying domain of times for the entire database [Snodgrass 1987].

However, its semantics may be confounded when various situations are considered. For instance, let us assume: <α, β> is a time interval of an existing tuple, <γ, δ> is a time interval to be deleted on the existing tuple, and <α, β> overlaps <γ, δ> as shown in Figure 5.1. Both time intervals are assumed to be closed at their lower end, shown by "|", and open at their upper end, indicated by ")". The intervals are highlighted by dashed lines.

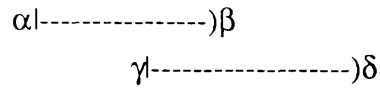


Figure 5.1 <α, β> overlaps <γ, δ>

According to the semantics of TQuel [Snodgrass 1987], for the delete operation, the actual deleted interval is <γ, β>, and we should add back another tuple with a time interval <α, γ>. There is a predicate statement for two such intervals by the definition of the Before predicate:

Before(α, γ) ∧ Before(β, δ) ∧ Before(γ, β)

However, according to the definition of the Before predicate,

Before = $\{t_i, t_j \mid t_i$ is less than or equal to $t_j\}$,

four other interval relationships can be obtained as shown in Figure 5.2.

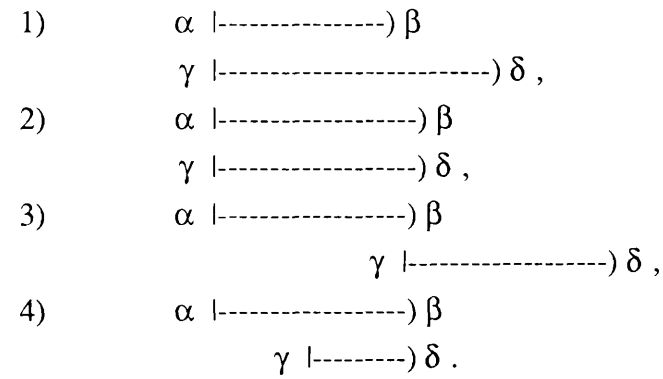


Figure 5.2 The Four Other Overlapping Relationships for The Predicates Before(α, γ) ∧ Before(β, δ) ∧ Before(γ, β)

In case 1 the two intervals have the same start time, but δ is later than β. For case 2 the two time intervals are exactly the same. In case 3 the upper bound β in the first interval is equal to the lower one γ in the second interval. The two upper bounds of intervals are the same in case 4, but α is before γ.

There is no need to add anything back for the cases 1), and 2). For the case 3), the delete operation deletes nothing, because the two intervals are only overlapped at the bounds. However, for the case 4), the actually deleted interval is $<\gamma, \delta>$, and the tuple with a time interval $<\alpha, \gamma>$ should be added back. The results are quite different. It is obvious that the definition of Before predicate does not give a unique result. It can confuse the semantics to apply such a definition to the temporal tuple calculus of TQuel.

Once we change the Before predicate instead as a linear order "<" predicate on integer time values, i.e., one event is *before* a second event if the time value of the first, expressed as an integer or real value, is less than (<) the time value of the second. and add another linear order predicate, Equal (=), to state one event is *equal* to second event if the time value of the first is equal to (=) the time value of the second:

Before(α, β) ::= $\alpha < \beta$

Equal(α, β) ::= $\alpha = \beta$

α, β : integers,

no confusion is made for the example above. For example, the predicate

Before (α, γ) $\wedge$ Before (β, δ) $\wedge$ Before (γ, β)

can only explain the case of $<\alpha, \beta>$ overlapping $<\gamma, \delta>$:

α|----------)β

γ|----------)δ ,

while the other four overlapping situations are presented as the following predicates:

1) Equal (α, γ) $\wedge$ Before (β, δ),

2) Equal (α, γ) $\wedge$ Equal (β, δ),

3) Equal (γ, β),

4) Before (α, γ) $\wedge$ Equal (β, δ).

We take this definition as the new underlying semantics for the predicates in TQuel.

5.1.2 The new definitions for temporal constructors and predicate operators

It is necessary to redefine the temporal constructors, the predicate operators, and the auxiliary functions with new definitions of Before and Equal predicates. Because the two new predicates are orthogonal, the old definitions [Snodgrass 1987] can be directly replaced with them[1]:

1) Auxiliary functions on integers (First, Last) or tuple variables (event, interval):

First (α, β) = α , (if *Before* (α, β) $\vee$ *Equal* (α, β)) ;

First (α, β) = β , (otherwise) ;

Last (α, β) = β , (if *Before* (α, β) $\vee$ *Equal* (α, β)) ;

Last (α, β) = α , (otherwise) ;

event (t)=<t_{at} , t_{at} > ;

interval (t)=<t_{from} , t_{to} > .

2) Temporal constructors:

beginof (<α, β>)=<α, α>

endof (<α, β>)=<β, β>

overlap (<α, β>,<γ, δ>)=<*Last* (α, γ), *First* (β, δ)>

extend (<α, β>,<γ, δ>)=<*First* (α, γ), *Last* (β, δ)>

assume: the intervals of the **overlap** function and the **extend** function do indeed overlap, i.e., we have constraints for **overlap** and **extend** constructors:

(*Before* (γ, β) $\vee$ *Equal* (γ, β)) $\wedge$ (*Before* (α, δ) $\vee$ *Equal* (α, δ))

3) Temporal predicate operators

precede (<α, β>,<γ, δ>) =*Before* (β, γ) $\vee$ *Equal* (β, γ)

---

[1] It can be illustrated that the new definitions can take the place of the old definition by checking the overlapped intervals' predicates.

$$overlap\ (<\alpha,\ \beta>,<\gamma,\ \delta>) = (Before\ (\alpha,\ \delta) \vee Equal\ (\alpha,\ \delta)) \wedge (Before\ (\gamma,\ \beta) \vee Equal\ (\gamma,\ \beta))$$

$$equal\ (<\alpha,\ \beta>,<\gamma,\ \delta>) = Equal\ (\alpha,\ \gamma) \wedge Equal\ (\beta,\ \delta)$$

5.2 Allen's Method of Representing the Relationships between Temporal Intervals

In the paper of [Snodgrass 1987], temporal relations were divided into event relations and interval relations. For event relations, which consist of tuples representing instantaneous occurrences, the time attribute contains a single time value (at). For interval relations, which consist of tuples representing a state valid over a time interval, the attribute contains two time values delimiting the interval (from, to). The model of TDB presents both to the user through the **valid at** and **valid from . . . to** clauses. To specify the derived time attributes, TQuel employs two unary prefix temporal constructors, **beginof** and **endof**, to return a single time value and employs two binary infix temporal constructors, **overlap** and **extend**, to return an interval value (an ordered pair of integers). TQuel also employs three binary infix temporal predicate operators, **precede**, **overlap** and **equal** to specify the relationships between two relative intervals by returning a Boolean value.

However, there are two problems with such a presentation:

1) As stated in Section 2.1, a model of time based on points on the real line does not correspond to the intuitive notion of time. A time point <t> would not be decomposable, while time intervals are decomposable;

2) There are only three temporal predicate operators representing the relationships between temporal intervals. However, as discussed in [Allen 1983], there are at least thirteen relationships between two overlapped intervals. Therefore, a multi-semantic

specification should be applied to one predicate operator. This would cause semantic problems.

A definition of interval relationships in [Allen 1983] can be applied to solve such problems. A time point <t> was informally taken as a very small interval $<t^-, t^+>$. Time model was considered as consisting of a fully ordered set of time points, and then an interval is an ordered pair of points with the first point *less than* the second.

Under such a time model, interval relationships are classified into seven relations. Considering the inverses of these relations, there are a total of thirteen ways in which an ordered pair of intervals can be related in Figure 5.3.

Relation	Symbol	Relation for Inverse	Symbol for Inverse
A before B	<	A after B	>
A meets B	m	A met by B	mi
A overlaps B	o	A overlapped by B	oi
A starts B	s	A started by B	si
A finishes B	f	A finished by B	fi
A during B	d	A contains B	di
A equal B	=	A equal B	=

Figure 5.3 Allen's Interval Relations

(A, B are relative intervals)

Assume: the relative intervals are A<α , β>, and B<γ , δ>. Then the overlap situations can be described as Figure 5.4.

Relation	Equivalent relations on endpoints	Pictorial example	Relation	Equivalent relations on endpoints	Pictorial example
A < B	β < γ	α|-----)β γ|------)δ	A > B	α > δ	α|-----)β γ|-----)δ
A m B	β = γ	α|-----)β γ|------)δ	A mi B	α = δ	α|-----)β γ|------)δ
A o B	α < γ, β < δ, γ < β	α|------)β γ|---------)δ	A oi B	α > γ, β > δ, δ > α	α|------)β γ|-------)δ
A s B	α = γ, β < δ	α|-----)β γ|----------)δ	A si B	α = γ, β > δ	α|----------)β γ|-----)δ
A f B	α > γ β = δ	α|-----)β γ|----------)δ	A fi B	α < γ β = δ	α|----------)β γ|-----)δ
A d B	α > γ β < δ	α|----)β γ|----------)δ	A di B	α < γ β > δ	α|----------)β γ|----)δ
A = B	α = γ β = δ	α|----------)β γ|----------)δ			

Figure 5.4 Allen's Interval Relations with Pictorial Examples

According to the definition in [Snodgrass 1987], three temporal predicate operators, **precede**, **overlap**, and **equal** can almost cover Allen's thirteen interval relationships (Figure 5.5). However, Allen's interval relations can be uniquely identified. One exclusively relates to one overlapping situation. A relation uniquely implies a temporal predicate, and the relative temporal predicate can only imply that relation (see Figure5.11 The actual Modified Intervals). No semantics are overlapped. The interval temporal constructors, **overlap** and **extend** can also be classified with Allen's thirteen interval relations (Figure 5.6).

Snodgrass Allen's	precede	overlap	equal	Snodgrass Allen's	precede	overlap	equal
A < B	α|---)β γ|----)δ			A > B	undefined		
A m B	α|----)β γ|----)δ	α|----)β γ|----)δ		A mi B	undefined	α|----)β γ|-----)δ	
A o B		α|------)β γ|-------)δ		A oi B		α|------)β γ|-------)δ	
A s B		α|-----)β γ|---------)δ		A si B		α|--------)β γ|----)δ	
A f B		α|---)β γ|--------)δ		A fi B		α|-------)β γ|-----)δ	
A d B		α|---)β γ|---------)δ		A di B		α|--------)β γ|----)δ	
A = B		α|--------)β γ|--------)δ	α|--------)β γ|--------)δ	undefined		α|)β γ|)δ	α|)β γ|)δ

Figure 5.5 Snodgrass' Temporal Operators and Allen's Interval Relations

In Figure 5.5 Snodgrass' **precede** operator can be stated as A **before** B, (A < B), and A **meet** B (A m B). The **overlap** operator can be defined as more than nine relationships of interval times (except one event relationship), while Allen's A **equal** B relation can be described as either **overlap** or **equal** in Snodgrass' definitions. Two classifications are compared according to their intervals' overlap situations. One situation (event situation) which can be included in Snodgrass' **overlap** or **equal** cases was not defined in Allen's interval relations. It is needed to wholly define overlap situations.

Snodgrass' Allen's	A overlap B	A extend B	Snodgrass' Allen's	A overlap B	A extend B
$\beta < \gamma$ A < B	α|-----)β γ|—)δ undefined null		$\alpha > \delta$ γ|—)δ A > B	α|-----)β undefined null	
$\beta = \gamma$ A m B	α|-----)β γ|—)δ <β, β>	α|------)β γ|——)δ <α, δ>	$\alpha = \delta$ A mi B	α|------)β γ|—)δ <α, α>	α|------)β γ|——)δ <γ, β>
$\alpha < \gamma$, $\gamma < \beta$, $\beta < \delta$ A o B	α|--------)β γ|———)δ <γ, β>	α|--------)β γ|———)δ <α, δ>	$\alpha > \gamma$, $\delta > \alpha$ $\beta > \delta$ A oi B	α|------)β γ|———)δ <α, δ>	α|------)β γ|———)δ <γ, β>
$\alpha = \gamma$, $\beta < \delta$ A s B	α|--------)β γ|————)δ <α, β>	α|--------)β γ|————)δ <γ, δ>	$\alpha = \gamma$, $\beta > \delta$ A si B	α|----------)β γ|——)δ <γ, δ>	α|---------)β γ|——)δ <α, β>
$\alpha > \gamma$, $\beta = \delta$ A f B	α|------)β γ|————)δ <α, β>	α|------)β γ|———)δ <γ, δ>	$\alpha < \gamma$, $\beta = \delta$ A fi B	α|---------)β γ|—)δ <γ, δ>	α|----------)β γ|——)δ <α, β>
$\alpha > \gamma$, $\beta < \delta$ A d B	α|------)β γ|————)δ <α, β>	α|------)β γ|————)δ <γ, δ>	$\alpha < \gamma$, $\beta > \delta$ A di B	α|----------)β γ|——)δ <γ, δ>	α|----------)β γ|——)δ <α, β>
$\alpha = \gamma$, $\beta = \delta$ A = B	α|----------)β γ|————)δ <α, β>	α|----------)β γ|————)δ <α, β>	$\alpha = \delta$, $\beta = \gamma$ undefined	αl)β γl)δ <α, β>	αl)β γ l)δ <α, β>

Figure 5.6 The Temporal Constructors with Allen's Interval Relations

In Figure 5.6, we not only explain the equivalent relation and pictorialize overlap situations of two relative intervals, but also state the results of Snodgrass' temporal constructors. For instance,

A **overlap** B,

if $\alpha < \gamma$, $\beta < \delta$, and $\gamma < \beta$ (A o B),

then the derived result of overlap temporal constructor is <γ , β>;

while if $\alpha > \gamma$, $\beta > \delta$, and $\delta > \alpha$ (A oi B), for the same constructor,

then the result is <α , δ>.

The result is absolutely different. Hence, the definitions of Snodgrass' temporal constructors are not clear enough in semantics. We use Allen's interval relations to classify Snodgrass' time relationships in Section 5.3. Such a classification causes no confusion in semantics.

5.3 The Modification Statements

5.3.1 Modification statements of Quel

Quel permits three commands: **replace**, **delete**, and **append**, which are update operations. The syntax of the modification statements is nearly identical to that of queries. Range statements have the same form and interpretation. The modification statements have the same basic form as retrieve statements:

Command Result-Name (Target-List)

where Qualification

For the **append** command, "Result-Name" must be the name of some existing relation, onto which qualifying tuples will be appended. For the **replace** (and **delete**) command, "Result-Name" must be a tuple variable which, through the qualification, identifies the tuples to be modified. The Target-List must contain explicitly (or by default) the existing Domain-Names for the relation being changed (the **delete** command has no Target-List).

5.3.1.1 The tuple calculus semantics for Quel modification statements

The material in this section is reproduced from [Snodgrass 1987].

The skeletal Quel append statement,

append to R $(t_{i_1}.D_{j_1}, \ldots, t_{i_r}.D_{j_r})$

where ψ

has the following tuple calculus semantics:

$$R' = R \cup \{u^{(r)} \mid (\exists t_1) \ldots (\exists t_k)(R_1(t_1) \wedge \ldots \wedge R_k(t_k)$$
$$\wedge (\forall l)\ (1 \le l \le r.\ u[l] = t_{i_l}[j_l])$$
$$\wedge \psi')\}$$

R' is combined by the original relation R and the set being appended which may contain tuples already in R.

The delete statement is

range of t_1 **is** R_1

. . . .

range of t_k **is** R_k

range of s **is** R

delete s

where ψ

and the relative tuple calculus statement is:

$$R' = \{s^{(r)} \mid (\exists t_1) \ldots (\exists t_k)(\exists s)(R_1(t_1) \wedge \ldots \wedge R_k(t_k)$$
$$\wedge \neg\psi')\}$$

The Quel replace statement

range of t_1 **is** R_1

. . . .

range of t_k **is** R_k

range of s **is** R

replace $s(t_{i_1}.D_{j_1}, \ldots, t_{i_r}.D_{j_r})$

where ψ

has the following tuple calculus semantics:

$$R' = \{u^{(r)} \mid (\exists t_1) \ldots (\exists t_k)(\exists s)(R_1(t_1) \wedge \ldots \wedge R_k(t_k)$$
$$\wedge ((u = s \wedge \neg\psi')$$
$$\wedge ((\forall l)\ (1 \le l \le r.\ u[l] = t_{i_l}[j_l]) \wedge \psi')))\}$$

Note that the second line is very similar to the tuple calculus semantics of the Quel delete statement, and the third line is identical to the semantics of the append statement. This means that a replace modification in Quel can be considered as one deletion followed by one append modification.

5.3.1.2 Examples for Quel modification statements

Some examples which are taken from the same relation as shown in Example 4.1.1 are given as below to show the formation of each modification command.

Example 5.3.1

Query: All information about the borrowers who live in Kelvin Street are added to the relation Kelvin.

```
range of b is BOOK-LOAN
append to Kelvin(Name = b.Name, Book# = b.Book#, Isbn = b.Isbn,
                 Date-due-back = b.Date-due-back)
where b.Address = "Kelvin"
```

Example 5.3.2

Query: Change Peter's address to Hillhead Street.

```
range of b is BOOK-LOAN
replace b( b.Address = "Hillhead St.")
where b.Name = "Peter"
```

Example 5.3.3

Query: Delete Book# 00030, because it was lost.

```
range of b is BOOK-LOAN
delete b
where b.Book# = "00030"
```

5.3.2 The modification statements for interval relations in TQuel

The literature [Snodgrass 1987] presented the semantics of the three TQuel modification statements (**append**, **delete**, and **replace**). Since TQuel is a strict superset of Quel, the relative TQuel modification statements and their calculus semantics (see Appendix C) are given by examining the tuple calculus semantics of the analogous Quel statements. Particularly, the replace modification was considered as a delete statement followed by an append statement as in Quel. However, according to such a semantics of modification, the modification operation, especially the replace modification, can only operate on a single relational tuple, but neither coalesced tuples nor disjoint tuples which are overlapped by the derived time intervals[1]. Such a problem will be argued by examples. Then, the conclusion is that although replace operation can roughly be thought of as a deletion followed by an append operation, it cannot be simply considered as a combination of them. We will discuss the resulting modification operations through the Allen's interval relationships and the new definitions of Before and Equal predicates as stated in Section 5.1 and Section 5.2 to make a foundation for the new semantics. Finally, according to such discussions, a new semantics of modification statements for interval relations will be presented. In the next subsection, the new semantics of modification statements for event relations will be presented using the same approach.

5.3.2.1 The problems with the replace statements

The representation of replace statements in [Snodgrass 1987] is semantically incomplete. First, the **when** clause is not necessary for the replace operation. Second, there are redundant predicate statements in the tuple calculus, and this will cause a

---

[1] This means that the replace operation cannot take place on tuples: $u_1, u_2, \ldots, u_n$, which have the same primary key, and the valid time of u_1 is $<t_1^-, t_1^+>, \ldots,$ the valid time of u_n is $<t_n^-, t_n^+>$, $t_i^+ \leq t_{i+1}^-$ (i=1, 2, . . .,n), and the derived interval $<\Phi_V, \Phi_\chi>$ overlaps $<t_1^-, t_n^+>$.

semantic error. Third, the tuple calculus of replace cannot work properly on multiple tuples which are overlapped by the derived time intervals. We will show these problems step by step by examples. An example of a temporal relation, Faculty, is shown in Figure 5.7. This is taken from [Snodgrass 1987]. In the temporal relation, tuples are assumed to be coalesced, in that tuples with identical values for the explicit attributes neither overlap nor are adjacent in time.

Faculty (Name, Rank):

		valid time		transaction time	
Name	Rank	from	to	start	stop
Jane	Assistant	9-71	12-76	9-71	∞
Jane	Associate	12-76	11-80	12-76	∞
Jane	Full	11-80	∞	10-80	∞
Merrie	Assistant	9-71	12-82	8-77	∞
Merrie	Associate	12-82	∞	12-82	∞
Tom	Associate	9-75	∞	8-75	10-75
Tom	Assistant	9-75	12-80	10-75	∞
Tom	Associate	12-80	∞	11-80	∞

Figure 5.7 A Temporal relation

1) First, we examine a simple TQuel replace statement.

Example 5.3.4 Merrie is promoted to full professor. The replace modification is executed in July 1988.

range of f **is** Faculty

replace f (Rank = "Full")

valid from "now" **to end of** f

where f.Name = "Merrie"

when f **overlap** "now"

as of "now"

The underlined clauses are default clauses. This is a very simple query for changing Merrie's rank from Associate Professor to Full Professor. It is executed on a

snapshot relation which is currently valid on both valid time and transaction time dimensions.

According to the semantics of modification statements in [Snodgrass, 1987], there is the following tuple calculus,

$$
\begin{aligned}
R' = \{u^{(2+4)} \mid (\exists f)\ (&\text{Faculty}\ (f) \\
&\wedge f[6] = \infty \wedge u[1] = f[1] \wedge u[2] = f[2] \wedge u[3] = f[3] \\
&\wedge u[4] = f[4] \wedge u[5] = f[5] \\
&\wedge ((\neg(\text{Affected} \wedge u[6] = f[6]) \vee (\text{Affected} \wedge u[6] = 7/88)))\} \\
\cup \{u^{(2+4)} \mid (\exists f)\ (&\text{Faculty}\ (f) \\
&\wedge f[6] = \infty \wedge u[1] = f[1] \wedge u[2] = f[2] \\
&\wedge \text{Affected} \wedge (C_1^{\,d} \vee C_2^{\,d} \vee C_3^{\,d} \vee C_4^{\,d}) \\
&\wedge u[5] = 7/88 \wedge u[6] = \infty)\} \\
\cup \{u^{(2+4)} \mid (\exists f)\ (&\text{Faculty}\ (f) \\
&\wedge f[6] = \infty \wedge u[1] = f[1] \wedge u[2] = \text{"Full"} \\
&\wedge u[5] = 7/88 \wedge u[6] = \infty \\
&\wedge f[1] = \text{"Merrie"} \wedge \text{Before}\ (f[3], 7/88) \wedge \text{Before}\ (7/88, f[4]) \\
&\wedge ((\exists f)\ (\text{Faculty}\ (f) \wedge f[1] = u[1] \wedge f[2] = u[2] \\
&\wedge (C_1^{\,a} \vee C_2^{\,a} \vee C_3^{\,a} \vee C_4^{\,a})) \\
&\quad \vee (\neg(\exists f)\ (\text{Faculty}\ (f) \wedge f[1] = u[1] \wedge f[2] = u[2] \\
&\quad \wedge u[3] = \Phi_V \wedge u[4] = \Phi_\chi)))\}
\end{aligned}
$$

where,

$\Phi_V = 7/88$

$\Phi_\chi = f[4]$

$$
\begin{aligned}
\text{Affected} = (&\text{Faculty}(f) \wedge f[1] = \text{"Merrie"} \wedge \text{Before}\ (f[3], 7/88) \\
&\wedge \text{Before}\ (7/88, f[4]) \\
&\wedge f[6] = \infty \wedge \text{Before}(f[3], \Phi_\chi) \wedge \text{Before}(\Phi_V, f[4]))
\end{aligned}
$$

$$
\begin{aligned}
C_1^{\,d} = (&\text{Before}(f[3], \Phi_V) \wedge \text{Before}(\Phi_V, f[4]) \wedge \text{Before}(f[4], \Phi_\chi) \\
&\wedge u[3] = f[3] \wedge u[4] = \Phi_V)
\end{aligned}
$$

C_2^d = (Before(Φ_V, f[3]) $\wedge$ Before(f[4], Φ_χ) $\wedge$ False)

C_3^d = (Before(Φ_V, f[3]) $\wedge$ Before(f[3],Φ_χ) $\wedge$ Before(Φ_χ, f[4])

$\wedge$ u[3] = Φ_χ $\wedge$ u[4] = f[4])

C_4^d = (Before(f[3], Φ_V) $\wedge$ Before (Φ_χ, f[4])

$\wedge$ ((u[3] = f[3] $\wedge$ u[4] = Φ_V) $\vee$ (u[3] = Φ_χ $\wedge$ u[4] = f[4]))

C_1^a = (Before(f[3],Φ_V) $\wedge$ Before(Φ_V, f[4]) $\wedge$ Before (f[4], Φ_χ)

$\wedge$ u[3] = f[4] $\wedge$ u[4] = Φ_χ)

C_2^a = (Before(Φ_V, f[3]) $\wedge$ Before(f[4], Φ_χ)

$\wedge$ ((u[3] = Φ_V $\wedge$ u[4] = f[3]) $\vee$ (u[3] = f[4] $\wedge$ u[4] = Φ_χ)))

C_3^a = (Before(Φ_V, f[3]) $\wedge$ Before(f[3],Φ_χ) $\wedge$ Before(Φ_χ, f[4])

$\wedge$ u[3] = Φ_V $\wedge$ u[4] = f[3])

C_4^a = (Before(f[3], Φ_V) $\wedge$ Before(Φ_χ, f[4]) $\wedge$ False)

Here, the underlying tuple participating in the query is the fifth tuple in Figure5.7. It is handled by the clauses C_1^d and C_1^a. In C_1^a, u[3]=f[4], u[4]=Φ_χ=f[4]; while f[4]=∞ in the relative tuple. It seems impossible for a valid tuple having the same valid from time and valid to time. We note that if a tuple has a right bound time value of ∞, that means that this tuple is available at the moment (now) till some moment in the future. So the domain for ∞ should include both *now* and some moment in the future, ∞. Then the infinite time value ∞ is a binary-state variable:

∞ = {current transaction id, ∞}.

We also note that the left bound of an interval cannot have an infinite value, ∞ (see Section 3.4 for detail). Therefore, in C_1^a,

u[3] = current transaction id,

u[4] = ∞.

Then the result of replace operations will be as in Figure 5.8. The transaction stop time of the fifth tuple in the relation has been changed.

Name	Rank	valid time		transaction time		
		from	to	start	stop	
Jane	Assistant	9-71	12-76	9-71	∞	
Jane	Associate	12-76	11-80	12-76	∞	
Jane	Full	11-80	∞	10-80	∞	
Merrie	Assistant	9-71	12-82	8-77	∞	
Merrie	Associate	12-82	∞	12-82	7-88	
Tom	Associate	9-75	∞	8-75	10-75	
Tom	Assistant	9-75	12-82	10-75	∞	
Tom	Associate	12-80	∞	11-80	∞	Set 1
Merrie	Associate	12-82	7-88	7-88	∞	Set 2
Merrie	Full	7-88	∞	7-88	∞	Set 3

Figure 5.8 The result of replace operation

There are two semantic problems with the tuple calculus stated above.

A. There is an useless predicate:

$\vee (\neg(\exists f)$ (Faculty (f) $\wedge$ f[1] = u[1] $\wedge$ f[2] = u[2] $\wedge$ u[3] = $\Phi_V \wedge$ u[4] = Φ_χ)

in the calculus. The replace tuple calculus was combined by three sets. The first set processes the deletion to deal with all tuples in past historical and current historical relations of R. For the tuples in past historical relations and those tuples which are in the current historical relation of R, but not **Affected**, the operation makes no change on them; for those **Affected** tuples in the current historical relation of R, the operation effectively removes them by setting their stop time to *current transaction id.*

The second set processes with the existing tuples which only partially should be deleted. Those portions that should not have been deleted are added back.

The third set is *exactly* a copy of the append operation. It appends the tuples with those portions which have not existed in the valid interval; and it appends the new tuples which *have not existed* during the entire valid time as the predicates stated in the last two lines.

However, the replace operation cannot replace any non-existent tuple. That is, it is *not necessary* to append the tuples which have not existed during the valid interval to the database. Therefore, the predicate

$\vee (\neg(\exists f)$ (Faculty (f) $\wedge$ f[1] = u[1] $\wedge$ f[2] = u[2] $\wedge$ u[3] = $\Phi_\nu \wedge$ u[4] = Φ_χ)

is useless in this tuple calculus and should be cut.

B. The **when** clause is not necessary in the replace operation. In Example 5.3.4, the **when** clause, **when** f **overlap** "now", specifies the temporal relationship of tuples participating in the derived relation. It seems to be working properly and selecting the valid tuples on the current active snapshot relation by the default predicate, Γ_τ:

Before(f[3], 7/88) $\wedge$ Before(7/88, f[4])

which is issued in all three sets of statements, specifying in the **Affected** clause.

However, suppose that the **valid** clause is changed into

valid from ν **to** χ

$\nu = \{ \Phi_\nu \mid \Phi_\nu \neq$ current transaction id $\}$;

$\chi = \{ \Phi_\chi \mid \Phi_\chi > \Phi_\nu \}$.

then, the **when** default clause cannot work properly, because it cannot retrieve the derived tuples from the current historical relation, except the current active snapshot relation. Only when the predicates are changed into

Before(f[3], Φ_χ) $\wedge$ Before(Φ_ν, f[4]) ,

i.e., the **when** clause is changed into

when f **overlap** $<\Phi_\nu, \Phi_\chi>$,

will tuple calculus retrieve the valid tuples on implicit valid time attributes.

However, such predicates have been issued by the **valid** clause and embedded in C_1^d , ..., C_4^d , C_1^a , ..., C_4^a , and the **Affected** clauses.

On the other hand, extending the domain of ν to include the current transaction id (now):

$\nu = \{ \Phi_\nu \mid$ any integer time value$\}$,

the when default clause can also be equivalently changed into the form

when f **overlap** <Φ_V, Φ_χ>

Φ_V = current transaction id

Φ_χ = **endof** f;

and the predicates are

Before(f[3], f[4]) $\wedge$ Before(current transaction id, f[4]).

These predicates have been stated in C_1^d, ..., C_4^d, C_1^a, ..., C_4^a, and **Affected** clauses as well. Clearly, it is not necessary to keep **when** clause in the **replace** statements. Therefore, the default **when** clause for **replace** should be that there is no **when** clause in it.

2) Now, we discuss the major semantic problem with overlapping tuples. The query as below is supported which has a derived interval time <9/71, 12/80> overlapping the first three tuples in the relation (Figure 5.7),

Example 5.3.5 **range of** f **is** Faculty

replace f (Rank = "Assistant")

valid from " September 1971" **to** "December 1980"

where f.Name = "Jane"

In the query, the derived interval <9/71, 12/80> overlaps three intervals in the relation: <9/71, 12/76>, <12/76, 11/80>, and <11/80, ∞> for the keyword "Jane" as shown in Figure 5.9.

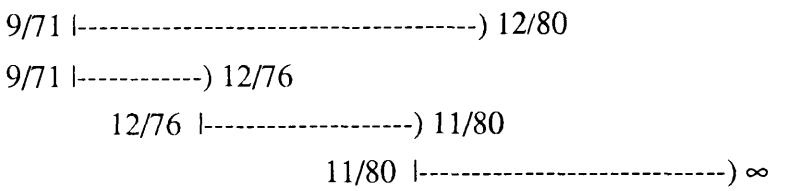


Figure 5.9 Intervals Overlapping

For these three intervals, the actual tuples replaced are really different. According to the semantics of replace statement in [Snodgrass 1987] (see Appendix C), the replace operation is deletion followed by an append operation, then we have,

1) for <9/71, 12/80> overlapping <9/71, 12/76>, the tuple with interval <9/71, 12/76> to be deleted, a redundant tuple with interval <9/71, 9/71> to be added back[1], and the new tuple with interval <12/76, 12/80> to be appended;

2) for <9/71, 12/80> overlapping <12/76, 11/80>, the tuple with interval <12/76, 11/80> to be deleted, appending two new tuples with intervals <9/71, 12/76> and <11/80, 12/80>;

3) for <9/71, 12/80> overlapping <11/80, ∞>, the tuple with interval <11/80, ∞> to be deleted, the tuple with interval <12/80, ∞> to be added back, and the new tuple with interval <9/71, 11/80> to be appended; then a new derived historical relation with redundant tuples is shown in Figure 5.10.

		valid time		transaction time			
Name	Rank	from	to	start	stop		
Jane	Assistant	9-71	12-76	9-71	7-88		
Jane	Associate	12-76	11-80	12-76	7-88		
Jane	Full	11-80	∞	10-80	7-88		
Merrie	Assistant	9-71	12-82	8-77	∞		
Merrie	Associate	12-82	∞	12-82	∞		
Tom	Associate	9-75	∞	8-75	10-75		
Tom	Assistant	9-75	12-80	10-75	∞		
Tom	Associate	12-80	∞	11-80	∞		Set 1
Jane	Assistant	9-71	9-71	7-88	∞	a.	
Jane	Full	12-80	∞	7-88	∞		Set 2
Jane	Assistant	12-76	12-80	7-88	∞	b.	
Jane	Assistant	9-71	12-76	7-88	∞	c.	
Jane	Assistant	11-80	12-80	7-88	∞	c.	
Jane	Assistant	9-71	11-80	7-88	∞	d.	Set 3

Figure 5.10 The Result of Replace Operation

Note: 1) we assume the transaction took place in July 1988;

[1] This is due to the Before predicate being ambiguous.

2) the results of replace operation are combined by three sets of tuples;

3) a. this tuple is as a result for Case C_1^d, and can be omitted when Case C_2^d is chosen. This is due to the predicate Before being ambiguous;

b. the result for the tuple with valid time interval <9/71, 12/76>, Case C_1^a was chosen; if the case C_2^a is chosen, there will be a redundant tuple with the valid interval <9/71, 9/71> in Set three.

c. the result for the tuple with valid time interval <12/76, 11/80>, Case C_2^a was chosen;

d. the result for the tuple with valid time interval <11/80, ∞>, Case C_3^a was chosen;

and we can see that the tuples in the derived relation are not coalesced at all now. We have some duplicate tuples with the intervals as follows:

```
9/71 |) 9/71
          12/76 |----------) 12/80
9/71 |----------) 12/76
```

or:

```
9/71 |) 9/71
                    11/80 |-------) 12/80
9/71 |----------) 11/80
```

We have shown three semantic problems with the TQuel replace statement, and solved two of them, the redundant predicates and the **when** clause, by simply deleting them. However, we have not solved the third problem, the operation on multiple tuples. In the next section, a model to develop new semantics of modification statements in TQuel will be presented. This model is based on the approach discussed in Section 5.1 and Section 5.2 which is the classification of temporal predicates operators and constructors with Allen's time interval relationships.

5.3.2.2 The actual modified intervals

According to the discussions in Section 5.1 and Section 5.2, the modification operations can be classified by Allen's time interval relationships.

Assume Interval A, $<\alpha,\beta>$ as an interval of the relative existing tuple, Interval B, $<\gamma,\delta>$ as an interval of the tuple to be modified; and there are constraints: Before(α,β), and Before(γ,δ) for them.

Also suppose that, the interval with * means the interval actually added back;

the interval with ** means the interval deleted;

the interval with *** means the new replaced interval;

nil means nothing to be added back;

null means impossible operations.

Then, the actual modified intervals of derived tuples is presented in Figure 5.11.

symbols	pictorial example	predicates	modifications		
			append	delete	replace
A < B	α|----)β γ|----)δ	Before(β,γ)	* <γ,δ>	null	null
A > B	α|----)β γ|----)δ	Before(δ,α)	* <γ,δ>	null	null
A m B	α|-----)β γ|----)δ	Equal(β,γ)	* <γ,δ>	null	null
A mi B	α|----)β γ|------)δ	Equal(δ,α)	* <γ,δ>	null	null
A o B	α|------)β γ|--------)δ	Before(α,γ)∧Before(β,δ) ∧Before(γ,β)	* <β,δ>	* <α,γ> **<α,β>	* <α,γ> ** <α,β> *** <γ,δ>
A oi B	α|-------)β γ|--------)δ	Before(γ,α)∧Before(δ,β) ∧Before(α,δ)	* <γ,α>	* <δ,β> **<α,β>	* <δ,β> **<α,β> *** <γ,δ>
A s B	α|------)β γ|----------)δ	Equal(γ,α)∧Before(β,δ)	* <β,δ>	* nil **<α,β>	** <α,β> *** <γ,δ>
A si B	α|----------)β γ|------)δ	Equal(γ,α)∧Before(δ,β)	* nil	* <δ,β> **<α,β>	* <δ,β> ** <α,β> *** <γ,δ>
A d B	α|----)β γ|----------)δ	Before(γ,α)∧Before(β,δ)	* <β,δ> <γ,α>	* nil **<α,β>	** <α,β> *** <γ,δ>
A di B	α|----------)β γ|----)δ	Before(α,γ)∧Before(δ,β)	* nil	* <δ,β> * <α,γ> **<α,β>	*<α,γ>,<δ,β> **<α,β> *** <γ,δ>
A f B	α|--------)β γ|-------- --)δ	Before(γ,α)∧Equal(δ,β)	* <γ,α>	* nil **<α,β>	** <α,β> *** <γ,δ>
A fi B	α|----------)β γ|----)δ	Before(α,γ)∧Equal(δ,β)	* nil	* <α,γ> **<α,β>	* <α,γ> ** <α,β> *** <γ,δ>
A = B	α|----------)β γ|----------)δ	Equal(γ,α)∧Equal(δ,β)	* nil	* nil **<α,β>	** <α,β> *** <γ,δ>
undefined	α|)β γ|)δ	Equal(α,δ)∧Equal(γ,β)	* nil	* nil **<α,β>	** <α,β> *** <γ,δ>

Figure 5.11 The Actual Modified Intervals

It is considered that the replace operation simply deletes the relative existing tuples (with interval (α,β)), appends new replacing tuples (with interval (γ,δ)), and adds back the tuples with portions which should not have been deleted. The deletion here only fixes the ordinary tuple with a finite recording stop time value, but not really deletes it. The replace operation can be considered as three orthogonal set operations. Neither operations depend on another operation, nor results influences each other. Thus, these operations can be processed in parallel. Due to being orthogonal, the semantics of the three set operations are not overlapped. Such an approach ensures no redundancy for the derived relations. This will be taken as an underlying model to discuss the new semantics in Section 5.3.2.3.

5.3.2.3 The semantics of modification statements for intervals

1) Append

The skeletal TQuel append statement of intervals is:

range of t_1 **is** R_1

. . . .

range of t_k **is** R_k

append to R $(t_{i_1}.D_{j_1}, \ldots, t_{i_r}.D_{j_r})$

valid from ν **to** χ

where ψ

when τ

then the tuple calculus statement for interval append is redefined as the following form:

$$R' = R \cup \{u^{(r+4)} \mid (\exists t_1) \ldots (\exists t_k)(R_1(t_1) \wedge \ldots \wedge R_k(t_k)$$

$$\wedge(\forall l)\ (1 \le l \le r.\ u[l] = t_{i_l}[j_l])$$

$$\wedge u[r+3] = \textit{current transaction id} \wedge u[r+4] = \infty$$

$$\wedge \psi'$$

$$\wedge \Gamma_\tau$$

$$\wedge(\forall l)(1 \le l \le k.t_l\ [\text{stop}] = \infty)$$

$$\wedge((\exists\ s)(R(s)\wedge(\forall l)\ (1\le l\le r.\ s\ [l\ ]=u\ [l\ ])$$

$$\wedge\ (C_1{}^a \vee C_2{}^a \vee C_3{}^a \vee C_4{}^a \vee C_5{}^a)) \qquad **$$

$$\vee((\exists\ s)(\neg R(s)\wedge(\forall l)\ (1\le l\le r.\ s\ [l\ ]=u\ [l\ ])$$

$$\wedge u\ [r+1] = \Phi_V \wedge u\ [r+2] = \Phi_\chi))))\}$$

Note: ** identifies those modified calculus which are different with Snodgrass' representation.

where,

$C_1{}^a$ = (((Before(s[r+1], Φ_V) $\wedge$ Before(Φ_V,s[r+2])) $\vee$ Equal(s[r+1], Φ_χ))

$\wedge$ Before(s[r+2], Φ_V) $\wedge$ u[r+1]= s[r+2] $\wedge$ u[r+2]= Φ_χ)

$C_2{}^a$ = ((Before(Φ_χ, s[r+2]) $\wedge$ Before(s[r+1], Φ_χ)) $\vee$ Equal(Φ_χ,s[r+2]))

$\wedge$ Before(Φ_V,s[r+1]) $\wedge$ u[r+1]=Φ_V $\wedge$ u[r+2]= s[r+1])

$C_3{}^a$ = (Before(Φ_V,s[r+1]) $\wedge$ Before(s[r+2],Φ_χ)

$\wedge$((u[r+1] = s[r+2]$\wedge$u[r+2] = Φ_χ)$\vee$(u[r+1] = $\Phi_V\wedge$u[r+2] = s[r+1]))

$C_4{}^a$ = (((Before(s[r+1], Φ_V) $\vee$ Equal(s[r+1], Φ_V))

$\wedge$ (Before(Φ_χ,s[r+2]) $\vee$ Equal(Φ_χ,s[r+2]))

$\vee$ (Equal(s[r+1], Φ_χ) $\wedge$ Equal(Φ_V,s[r+2])))$\wedge$ False)

$C_5{}^a$ = ((Before(s[r+2], Φ_V) $\vee$ Equal(s[r+2], Φ_V)

$\vee$Before(Φ_χ,s[r+1]) $\vee$ Equal(Φ_χ,s[r+1]))

$\wedge$ u[r+1]= Φ_V $\wedge$ u[r+2]= Φ_χ)

The interval <s[r+1], s[r+2]> is an interval of the existing tuple, and <Φ_V, Φ_χ> is an interval to be modified. Due to the Before ($\le$) predicate being changed into Before (<) and Equal (=) two predicates, the classified predicates in Figure 5.11 are used. In Figure 5.11, there are simply three intervals actually added back after an append operation: <S[r+2], Φ_χ> and <Φ_V , S[r+1]> for overlapped intervals, and <Φ_V, Φ_χ> for unoverlapped intervals. According to these three intervals and the predicates, we can obtain five overlap situations between the tuples to be added and the tuples identical in the

explicit attributes that already exist during this valid interval as stated as C_1^a, C_2^a, C_3^a, C_4^a and C_5^a.

Because C_1^a, C_2^a, C_3^a, C_4^a and C_5^a denote the relationships of time intervals in append tuple calculus statement, and have been modified according to the new definition of Before predicate, such a tuple calculus statement makes no confusion in semantics. For instance, the overlap situation, <s[r+1], s[r+2]> **overlap** <Φ_ν, Φ_χ>, where Φ_ν < S[r+1] and Φ_χ < S[r+2], corresponds to C_2^a case, while the predicate Before(Φ_ν , S[r+1]) $\wedge$ Before(Φ_χ, S[r+2]) in C_2^a can only explain such a situation.

2) Delete

The TQuel delete statement of intervals is:

range of t_1 **is** R_1

. . . .

range of t_k **is** R_k

range of s **is** R

delete s

valid from ν **to** χ

where ψ

when τ

then the tuple calculus statement is:

$$R' = \{u^{(r+4)} \mid (\exists t_1) \ldots (\exists t_k)(\exists s)(R_1(t_1) \wedge \ldots \wedge R_k(t_k)$$

$$\wedge(\forall l)(1 \le l \le k . t_l\ [\text{stop}] = \infty)$$

$$\wedge(\forall l)\ (1 \le l \le r .\ u[l] = s[l]) \wedge u[r+1] = s[r+1]$$

$$\wedge u[r+2] = s[r+2] \wedge u[r+3] = s[r+3]$$

$$\wedge((\neg \text{Affected} \wedge u[r+4] = s[r+4])$$

$$\vee(\text{Affected} \wedge u[r+4] = \textit{current transaction id}))) \}$$

$$\cup \{u^{(r+4)} \mid (\exists t_1) \ldots (\exists t_k)(\exists s)(R_1(t_1) \wedge \ldots \wedge R_k(t_k)$$

$$\wedge(\forall l)(1 \le l \le k . t_l\ [\text{stop}] = \infty)$$

$\wedge(\forall l)\ (1 \le l \le r.\ u[l] = s[l]) \wedge$ Affected

$\wedge\ (C_1{}^d \vee C_2{}^d \vee C_3{}^d \vee C_4{}^d)$ **

$\wedge u[r+3] =$ *current transaction id* $\wedge u[r+4] = \infty)\}$

where,

Affected = $(R(s) \wedge \psi' \wedge \Gamma_\tau \wedge s[r+4] = \infty$

$\wedge(\text{Before}(s[r+1], \Phi_\chi) \wedge \text{Before}(\Phi_\nu, s[r+2]))$

$C_1{}^d = (((\text{Before}(\Phi_\nu, s[r+1]) \wedge \text{Before}(s[r+1], \Phi_\chi)) \vee \text{Equal}(\Phi_\nu, s[r+1]))$

$\wedge\ \text{Before}(\Phi_\chi, s[r+2]) \wedge u[r+1] = \Phi_\chi \wedge u[r+2] = s[r+2])$

$C_2{}^d = (((\text{Before}(s[r+2], \Phi_\chi) \wedge \text{Before}(\Phi_\nu, s[r+2])) \vee \text{Equal}(s[r+2], \Phi_\chi))$

$\wedge\ \text{Before}(s[r+1], \Phi_\nu) \wedge u[r+1] = s[r+1] \wedge u[r+2] = \Phi_\nu)$

$C_3{}^d = (\text{Before}(s[r+1], \Phi_\nu) \wedge \text{Before}(\Phi_\chi, s[r+2])$

$\wedge((u[r+1] = \Phi_\chi \wedge u[r+2] = s[r+2]) \vee (u[r+1] = s[r+1] \wedge u[r+2] = \Phi_\nu))$

$C_4{}^d = (((\text{Before}(\Phi_\nu, s[r+1]) \vee \text{Equal}(\Phi_\nu, s[r+1]))$

$\wedge(\text{Before}(s[r+2], \Phi_\chi) \vee \text{Equal}(s[r+2], \Phi_\chi))$

$\vee(\text{Equal}(s[r+1], \Phi_\chi) \wedge \text{Equal}(\Phi_\nu, s[r+2])))) \wedge$ False)

There are two sets in the tuple calculus statements of delete operation. One contains all tuples in past historical relations of *R* and all tuples in the current historical relation of *R* that are not **Affected**, that is, that do not satisfy the predicate in the **where** or **when** clauses or whose valid intervals do not overlap with the specified valid interval. Another set deals with the existing tuples that only partially should be deleted. Those portions that should not have been deleted are added back in the second set. The clauses $C_1{}^d$, $C_2{}^d$, $C_3{}^d$, and $C_4{}^d$ calculate the valid times of those tuples using the same approach in Append modification. Only two intervals are to be added back <S[r+1], Φ_ν>, and <Φ_χ, S[r+2]>. Unoverlapped intervals are not allowed in the delete operation and replace operation. This statement also deals with the **Affected** tuples, simply removing them by setting their stop time to current transaction identifier.

3) Replace

The TQuel replace statement of intervals is:

range of t_1 **is** R_1

. . . .

range of t_k **is** R_k

range of s **is** R

replace $s(t_{i_1}.D_{j_1}, \ldots, t_{i_r}.D_{j_r})$

valid from ν **to** χ

where ψ

then the tuple calculus statement for interval replacing has the following form:

$$
\begin{aligned}
R' = \{u^{(r+4)} \mid & (\exists t_1) \ldots (\exists t_k)(\exists s)(R_1(t_1) \wedge \ldots \wedge R_k(t_k) \\
& \wedge (\forall l)(1 \le l \le k . t_l\ [\text{stop}] = \infty) \\
& \wedge (\forall l)\ (1 \le l \le r .\ u[l] = s[l]) \wedge u[r+1] = s[r+1] \\
& \wedge u[r+2] = s[r+2] \wedge u[r+3] = s[r+3] \\
& \wedge ((\neg \text{Affected} \wedge u[r+4] = s[r+4]) \\
& \quad \vee (\text{Affected} \wedge u[r+4] = \textit{current transaction id}))) \} \\
\cup \{u^{(r+4)} \mid & (\exists t_1) \ldots (\exists t_k)(\exists s)(R_1(t_1) \wedge \ldots \wedge R_k(t_k) \\
& \wedge (\forall l)(1 \le l \le k . t_l\ [\text{stop}] = \infty) \\
& \wedge (\forall l)\ (1 \le l \le r .\ u[l] = s[l]) \wedge \text{Affected} \\
& \wedge (C_1{}^d \vee C_2{}^d \vee C_3{}^d \vee C_4{}^d) \qquad ** \\
& \wedge u[r+3] = \textit{current transaction id} \wedge u[r+4] = \infty) \} \\
\cup \{u^{(r+4)} \mid & (\exists t_1) \ldots (\exists t_k)(R_1(t_1) \wedge \ldots \wedge R_k(t_k) \\
& \wedge (\forall l)\ (1 \le l \le r .\ u[l] = t_{i_l}[j_l]) \\
& \wedge u[r+3] = \textit{current transaction id} \wedge u[r+4] = \infty \\
& \wedge \psi' \\
& \wedge (\forall l)(1 \le l \le k .\ t_l\ [\text{stop}] = \infty) \\
& \wedge ((\exists s)(R(s) \wedge (\forall l)\ (1 \le l \le r .\ s[l] = u[l]) \\
& \quad \wedge u[r+1] = \Phi_\nu \wedge u[r+2] = \Phi_\chi))) \} \qquad **
\end{aligned}
$$

where,

Affected = $(R\ (s) \wedge \psi' \wedge s\ [r+4\ ]=\infty$

$$\wedge(\text{Before}(s\ [r+1\ ], \Phi_\chi) \wedge \text{Before}(\Phi_\nu, s\ [r+2\ ]))$$

$C_1^{\ d}, \ldots, C_4^{\ d}$ clauses are as stated in delete statements. Comparing the replace statement in [Snodgrass 1987] (see Appendix C), we note that there is no need as in append statements to deal with the tuples which do not satisfy the **where** predicate or do not exist during the valid interval. However, in [Snodgrass, 1987] such tuples were handled. The semantics of replace statement was combined by those of delete and append statements directly.

Three sets are processed. The first two of them are like the statements in delete operation, the third is almost like those in the append statements, but just simply appends new replacing tuples with interval $<\Phi_\nu, \Phi_\chi>$ to derived relations.

Checking the same query as stated in Example 5.3.5, the historical relation with no redundant tuples can be obtained in Figure 5.12. The tuples are coalesced in the derived relation as well.

Faculty (Name, Rank):

		valid time		transaction time		
Name	Rank	from	to	start	stop	
Jane	Assistant	9-71	12-76	9-71	7-88	
Jane	Associate	12-76	11-80	12-76	7-88	
Jane	Full	11-80	∞	10-80	7-88	
Merrie	Assistant	9-71	12-82	8-77	∞	
Merrie	Associate	12-82	∞	12-82	∞	
Tom	Associate	9-75	∞	8-75	10-75	
Tom	Assistant	9-75	12-80	10-75	∞	
Tom	Associate	12-80	∞	11-80	∞	Set 1
Jane	Full	12-80	∞	7-88	∞	Set 2
Jane	Assistant	9-71	12-80	7-88	∞	Set 3

Figure 5.12 The New Result of Replace Operation

5.3.3 The modification statements for event relations

Events and intervals are quite similar semantically. A time point <t> can be informally taken as a very small interval <t^-, t^+>. TQuel presents both to the user through the **valid at** and **valid from . . . to** clauses in the retrieve statement. However, events and intervals are quite different in the tuple calculus statements, especially in the modification statements. The literature [Snodgrass 1987] did not present such statements for events. Fortunately, they are easy to derive. We represent the append, delete, and replace statements of events here, and assume that all relations to be operated are event relations. We give the relationships of events and the rules for event modifications first, then present the TQuel calculus statements for append, delete, and replace operations.

5.3.3.1 The temporal relationships between the existing tuple and the tuple to be modified

There are three relationships for events (**before**, **equal**, and **after**):

	1)	2)	3)
existing tuple	s[r+1] ∘	s[r+1] ∘	s[r+1] ∘
tuple to be modified	Φ_υ ∘	Φ_υ ∘	Φ_υ ∘
	$s[r+1] < \Phi_\upsilon$	$s[r+1] = \Phi_\upsilon$	$s[r+1] > \Phi_\upsilon$

Figure 5.13 The Relationships of Event Modification

5.3.3.2 The rules for event modifications

For any already existing event, we need not append anything. We only append the event which is not available at the time and will be valid after the operation. Deleting and replacing only process the event which is already valid and to be modified.

Therefore, for the append operation, we append nothing in the cases 1) and 2), but append a new tuple with valid time at Φ_υ to the most recent historical relation in the case

3); the existing tuples can be modified (deleted or replaced) in the cases 1) and 2), but cannot be modified at all in the case 3).

5.3.3.3 The TQuel calculus statements for **append**, **delete**, and **replace** operations

1) Append

The skeletal TQuel append statement of events is:

range of t_1 **is** R_1

. . . .

range of t_k **is** R_k

append to R $(t_{i_1}.D_{j_1}, \ldots, t_{i_r}.D_{j_r})$

valid at ν

where ψ

when τ

then the tuple calculus statement for event appending has the following form:

$$R' = R \cup \{u^{(r+3)} \mid (\exists t_1) \ldots (\exists t_k)(R_1(t_1) \wedge \ldots \wedge R_k(t_k)$$

$$\wedge(\forall l)\ (1 \le l \le r.\ u[l] = t_{i_l}[j_l])$$

$$\wedge u[r+2] = \textit{current transaction id} \wedge u[r+3] = \infty$$

$$\wedge \psi'$$

$$\wedge \Gamma_\tau$$

$$\wedge(\forall l)(1 \le l \le k.t_l\ [\text{stop}] = \infty)$$

$$\wedge((\exists s)(R(s) \wedge (\forall l)\ (1 \le l \le r.\ s[l] = u[l]) \wedge (C_1^a \vee C_2^a))$$

$$\vee((\exists s)(\neg R(s) \wedge (\forall l)\ (1 \le l \le r.\ s[l] = u[l]) \wedge u[r+1] = \Phi_\nu)))$$

$$))\}$$

where,

C_1^a = ((Before(s[r+1], Φ_ν) $\vee$ Equal (s[r+1], Φ_ν)) $\wedge$ False)

C_2^a = (Before(Φ_ν, s[r+1]) $\wedge$ u[r+1] = Φ_ν)

2) Delete

The TQuel delete statement of events is:

range of t_1 **is** R_1

. . . .

range of t_k **is** R_k

range of s **is** R

delete s

valid at ν

where ψ

when τ

the tuple calculus statement has the following form:

$$R' = \{u^{(r+3)} \mid (\exists t_1) \ldots (\exists t_k)(\exists s)(R_1(t_1) \wedge \ldots \wedge R_k(t_k)$$

$$\wedge(\forall l)(1 \le l \le k . t_l\ [\text{stop}] = \infty)$$

$$\wedge(\forall l)\ (1 \le l \le r .\ u\ [l\ ] = s\ [l\ ])\ \wedge u\ [r+1\ ] = s\ [r+1\ ] \wedge u\ [r+2\ ] = s\ [r+2\ ]$$

$$\wedge((\neg \text{Affected} \wedge u\ [r+3\ ] = s\ [r+3\ ])$$

$$\vee(\text{Affected} \wedge u\ [r+3\ ] = \textit{current transaction id}\ ))))\}$$

where,

$$\text{Affected} = (R\ (s) \wedge \psi' \wedge \Gamma_\tau \wedge s\ [r+3\ ] = \infty$$

$$\wedge(\text{Before}(s[r+1],\ \Phi_V)\ \vee\ \text{Equal}\ (s[r+1],\ \Phi_V)))$$

3) Replace

The TQuel replace statement of events is:

range of t_1 **is** R_1

. . . .

range of t_k **is** R_k

range of s **is** R

replace $s(t_{i_1}.D_{j_1}, \ldots, t_{i_r}.D_{j_r})$

valid at ν

where ψ

then the tuple calculus statement for event replacing has the following form:

$$R' = \{u^{(r+3)} \mid (\exists t_1) \ldots (\exists t_k)(\exists s)(R_1(t_1) \wedge \ldots \wedge R_k(t_k)$$

$$\wedge(\forall l)(1 \le l \le k.t_l\ [\text{stop}] = \infty)$$

$$\wedge(\forall l)\ (1 \le l \le r.\ u\ [l\ ] = s\ [l\ ]) \wedge u\ [r+1\ ] = s\ [r+1\ ] \wedge u\ [r+2\ ] = s\ [r+2\ ]$$

$$\wedge((\neg \text{Affected} \wedge u\ [r+3\ ] = s\ [r+3\ ])$$

$$\vee(\text{Affected} \wedge u\ [r+3\ ] = \textit{current transaction id}\ ))$$

$$)\}$$

$$\cup \{u^{(r+3)} \mid (\exists t_1) \ldots (\exists t_k)(R_1(t_1) \wedge \ldots \wedge R_k(t_k)$$

$$\wedge(\forall l)\ (1 \le l \le r.\ u\ [l\ ] = t_{i_l}[j_l\ ])$$

$$\wedge u\ [r+2] = \textit{current transaction id} \wedge u\ [r+3] = \infty$$

$$\wedge \psi'$$

$$\wedge(\forall l)(1 \le l \le k.t_l\ [\text{stop}] = \infty)$$

$$\wedge((\exists s)(R(s) \wedge (\forall l)\ (1 \le l \le r.\ s\ [l\ ] = u\ [l\ ])\ \wedge u\ [r+1\ ] = \Phi_V)))\}$$

where,

$$\text{Affected} = (R\ (s) \wedge \psi' \wedge s\ [r+3\ ] = \infty$$

$$\wedge(\text{Before}(s[r+1],\ \Phi_V) \vee \text{Equal}\ (s[r+1],\ \Phi_V)))$$

The two clauses, C_1^a and C_2^a, handle the various situations between the tuples to be appended and the tuples identical in the explicit attributes that already exist. C_1^a explains the cases 1) and 2) as stated above, and C_2^a explains the case 3). While the Affected clause states all tuples that satisfy the predicate in the **where** clauses and whose event valid time is before or equal to the specified valid time. These tuples can be modified by the delete and replace statements. There is no need to add any portions back for the event delete and replace statement. In the statement of event replace operation, the first set contains all tuples in past historical relations of R. The second set deals with the all tuples in the current historical relation of R that satisfy the predicate in the **where** clauses.

Chapter 6 Conclusion and Further Research

In this chapter the work of this dissertation is reviewed, in particular: the conceptual semantics of time, the classification of the different types of temporal database models and the comparison of temporal query languages, including improvement of the semantics of modification statements.

This dissertation has concentrated on the association of time with tuples and in the next section alternative approaches are considered. This is followed by further discussion of the relationship between entities and tuples. Finally, some outstanding problems are briefly discussed: temporal schema evolution, the integrity of time attributes and the implementation of temporal databases.

6.1 Work which has been done

The review of research about TDBMS in this dissertation has three emphases:

1) *The formulation of a semantics of time at the conceptual level.* This has been discussed in some depth in Chapter 2. A topology of time and types of time attributes were introduced. A new taxonomy for time attributes was presented: assertion time, event time, and recording time. They stamp the information in the real world at three different levels: user, event, and system, respectively. Because the event time of an entity in the historical chain can be changed into assertion time, in some sense, it can be treated as a kind of assertion time. Thus, in most TDBMSs, only two time attributes: event time (logical time) and recording time (physical time) are supported.

2) *The development of a model for TDBMS analogous to the relational model for snapshot databases.* Based on Snodgrass' classification, four kinds of databases are discussed in depth. The discussion notes some differences from the representation of the TDB model. Main contributions and differences with TDB model in Chapter 3 are:

- historical relation for most enterprises is an interval relation, but not a sequence of snapshot slices indexed by valid time.

- tuple no longer simply refers to an entity as in traditional relational databases. It refers to different level representations of an object: entity, entity state, observation of entity, and observation of entity state in different databases.
- domains for time attributes were discussed. The retroactive and proactive changes of temporal data are explained through the values of time attributes.
- an approach to merge temporal tuples without losing any temporal semantics was introduced to save more storage in the databases.

3) *The design of temporal query languages.* We have not presented a new temporal query language in this dissertation, but we have discussed a Quel-like temporal query language, TQuel, in some depth. Comparisons between TQuel and two other temporal query languages emphasized the need to take TQuel as the main discussed model. However, TQuel is not perfect in its semantics model. Thus, we centred the main discussion on TQuel's semantics for tuple calculus. The semantics problem for the *Before* predicate has been discussed and a solution presented to it is to define *Before* as a single valued predicate, not a bi-valued one. The classification for the relationships between overlapping intervals suggested an approach using temporal logic to classify the derived tuples in tuple calculus. Under such an approach, a new presentation for tuple modification calculus was proposed, not only for event relations, but also for interval relations.

6.2 Treating Time as a Component of Tuples

So far, the method to represent temporal database models introduced in this dissertation is embedding a temporal relation in a snapshot relation by appending two (or one) time attributes, each containing two (or one) time values, denoting intervals (or points) of logical time (and/or physical time). Such an approach incorporates temporal dimensions into the relational model at the *tuple level.* Several benefits accrue from such a representation:

1) The snapshot relational model can be used as the underlying model. The relational database model is simple and is based on the well-developed formalisms of set theory and predicate calculus; database models directly incorporating time are significantly more complex and are based on newer and less well-understood logics such as multiple transaction and temporal logics; and extensions involving aggregates and indeterminacy are easier to formulate in the standard model. Finally, a temporal database based on the relational model can be implemented directly on conventional relational DBMSs.

2) The language can be designed as a minimal extension, both syntactically and semantically, of one of the well-developed database languages. For instance, Quel is widely discussed in the literatures; it is particularly simple, but rather powerful; and it has a simple and well-defined semantics. TQuel is designed as a superset of Quel. All legal Quel statements are also valid TQuel statements, such statements have an identical semantics in Quel and TQuel when the time domain is fixed, and the additional constructs defined in TQuel to handle time have direct analogues in Quel.

3) Ease of formal manipulation and the promise of rapidly prototyping a temporal DBMS on top of a conventional snapshot DBMS.

However, treating time as a component of the tuples causes difficulties on semantics maintenance and performance improvement:

Time attributes in such a model belong to all time-varying attributes (i.e., to the whole tuple, not only to any one of them!). Changing values of any attribute will cause at least one new tuple to be generated. However, we cannot say at what time which attribute was changed according to a derived tuple. Hence, making time stamps only at tuple level is not clear enough for expressing every time-varying object in the relation.

Embedding time at the tuple level is reasonable from the logic point of view, but is ineffective from an operational point of view. With the method of appending time attributes to relations, all tuples belonging to the same relation will be treated as equivalent, regardless of the time aspect. The history of all the different actions, delete, update, insert, or create, will be treated as equals and each action will add one or more new tuples to the database. This will cause the database to grow very fast, unless the

database is static or semi-static. The result is that we will have performance and storage problems. A benchmark set of queries was run to study the performance of the prototype of such an approach [Ahn & Snodgrass 1986]. As expected, the performance rapidly deteriorated as information was added to the database. Access methods such as sequential scan, hashing, and ISAM all suffered. In addition, overflow chains increase exponentially due to all versions of a tuple sharing the same key.

Another approach to incorporating the temporal dimension into the relational model is embedding the time at the *attribute level* [Clifford & Tansel 1985, Segev & Shoshani 1987]. There are several reasons for treating time as a component of the attributes:

1) attributes vary over time in different ways. Different attributes may be measurable/recordable at different rates (the granularity may be different). There are time-dependent and time-independent (static) attributes. When a change occurs, it is generally to the value of an individual attribute, not to all attributes in a tuple.

2) the projection operator sometimes makes no sense or loses information in a relation where each tuple is time-stamped. For example, consider the relation Book-Location shown in Figure 6.1 (drawn from Figure 3.10), and consider how we "project" out the books on shelf.

			Event time		Recording time	
Book#	Isbn	Shelf	from	to	start	end
00023	0-999-99999-0	cb001	10/4/88	∞	12/4/88	∞
00012	0-12345-123-x	cb002	10/4/88	∞	12/4/88	18/10/88
00012	0-12345-123-x	cb002	20/5/88	∞	18/10/88	∞
00030	0-332-42233-1	cb004	10/4/88	∞	12/4/88	1/10/88
00030	0-332-42233-1	cb004	10/4/88	1/10/88	1/10/88	∞
00065	0-999-99999-0	cb001	26/6/88	∞	25/6/88	1/10/88
00065	0-999-99999-0	cb001	26/6/88	15/9/88	1/10/88	∞
00065	0-999-99999-0	cb003	15/9/88	∞	1/10/88	∞
00080	0-5656-5566-2	cb003	1/10/88	∞	1/10/88	∞

Figure 6.1 Book-Location Temporal Relation

If we simply project out the column, Book#, together with the temporal attributes, we get a temporal relation like the one shown in Figure 6.2.

Book#	Event time		Recording time	
	from	to	start	end
00023	10/4/88	∞	12/4/88	∞
00012	10/4/88	∞	12/4/88	18/10/88
00012	20/5/88	∞	18/10/88	∞
00030	10/4/88	∞	12/4/88	1/10/88
00030	10/4/88	1/10/88	1/10/88	∞
00065	26/6/88	∞	25/6/88	1/10/88
00065	26/6/88	15/9/88	1/10/88	∞
00065	15/9/88	∞	1/10/88	∞
00080	1/10/88	∞	1/10/88	∞

Figure 6.2 A Derived Relation

Note: assume we can derive a temporal relation (see Section 6.4 for detail).

We have lost relevant information from the relation. The tuples in the relation could not make sense, because we do not know which book is in which shelf and what the time intervals mean. The primary key for the relation in Figure 6.1 is the attributes, Book# and Shelf, together with temporal information. Without the attribute Shelf, such a projection result cannot explain the relationships among the tuples.

However, there are at least two important ramifications for such an approach. The first is that relations in the model are no longer in *First Normal Form*, since the domain for time-varying attributes is non-simple. Nested relational databases [Roth et al. 1988] may be necessary. The second is that with attributes differing along various time-related dimensions, the increasing complexity both of syntax and semantics can be imagined. It becomes necessary to define some underlying "basic" view of time for the database.

6.3 Operations for Entity, Entity State, Observation of Entity, Observation of Entity State and Tuple

The terminology of entity, entity state, observation of entity, and observation of entity state is independent of any specific logical data model, such as relational database. These concepts could be designed for other database models, like nested

relational databases [Roth et al. 1988] and the TSC model in [Segev & Shoshani 1987]. However, more semantics study is needed for them. For example, how about the traditional operations for them? Could they be retrieved using tuple calculus in temporal relational databases? How about the modification for them? Although we have developed the tuple calculus for temporal tuples and can retrieve an observation of an entity state (a tuple) properly, it is necessary to develop more semantics and syntax for other operations.

Entity, entity state, observation of entity, and observation of entity state are respectively represented as a single tuple in four distinct databases, and represent different sets of tuples in the temporal database. Using tuple calculus, we can handle an observation of an entity state as stated in the early chapters. However, the retrieval of entities in the database suffers from the lack of temporal semantics. Taking the database in Figure 6.1 as an example, to list out the entities (books) in the database, we issue a query:

range of a **is** Book-Location

retrieve (a.Book#)

a result will be as:

Book#
00023
00012
00030
00065
00080

but, we have no idea about the lifespans of entities. How to derive the valid intervals for each entity is still under study.

To retrieve the entity states of Book# 00065 as best known, a query is:

range of a **is** Book-Location

retrieve (a.Book#, a.Isbn, a.Shelf)

where a.Book# = "00065"

as of "now"

and the result is:

Book#	Isbn	Shelf	Event time		Recording time	
			from	to	start	end
00065	0-999-99999-0	cb001	26/6/88	15/9/88	1/10/88	∞
00065	0-999-99999-0	cb003	15/9/88	∞	1/10/88	∞

However, we cannot issue a query to retrieve the observations for an entity in the temporal database easily. An ideal approach is picking up all tuples for the same entity first, then according to the comments in Section 3.4.2, searching for the recording start times, if the value is the same for two tuples, then they are the same observation for this entity, but for different states of this entity. Clearly, tuple calculus is not satisfactory for such operations. New syntax for modify operations need to be developed as well.

6.4 Temporal Schema Evolution

We define the *temporal schema* as that part of the database schema related to time attributes which is hidden from the user and managed by the system. This temporal schema may evolve and the following example illustrates this evolution.

As stated in Chapter 3, for the following temporal query on a temporal database Book-Location (Figure 3.10), we have not presented a good enough mechanism to record the recording times, and the derived relation is a historical relation not a temporal one:

range of b **is** BOOK-LOCATION.temporal

retrieve into BOOK65(Book#=b.Book#, Isbn=b.Isbn, Shelf=b.Shelf)

where b.Book# = "00065"

when b **overlap** b

as of "20/10/88"

A derived relation (assume that the transaction was taken on 25th November) is

Book#	Isbn	Shelf	Event time	
			from	to
00065	0-999-99999-0	cb001	26/6/88	15/9/88
00065	0-999-99999-0	cb003	15/9/88	∞

However, there are three recording times which have not been captured:

25/11/88 -- the system time for the retrieve transaction;

20/10/88 -- the (**as of**) best known time for the query; and

1/10/88 -- the recording start time for the entity states of Book# 00065.

How can we create the mechanism? In [Snodgrass & Ahn 1985], the authors proposed an example which captured the third recording time 1/10/88. The derived relation is:

Book#	Isbn	Shelf	Event time		Recording time	
			from	to	start	end
00065	0-999-99999-0	cb001	26/6/88	15/9/88	1/10/88	∞
00065	0-999-99999-0	cb003	15/9/88	∞	1/10/88	∞

The result presented the original information about the observation of entity state of Book# 00065. However, such a representation does not satisfy the definition: recording time is automatically maintained (presented) by TDBMS after each transaction. To be satisfied with the definition, a semantics for the retrieve tuple calculus has been proposed in [Snodgrass 1987]:

$$
\begin{aligned}
R' = \{ u^{(r+4)} \mid\ & (\exists t_1) \ldots (\exists t_k)(R_1(t_1) \wedge \ldots \wedge R_k(t_k) \\
& \wedge (\forall l)\ (1 \le l \le r.\ u[l] = t_{i_l}[j_l]) \\
& \wedge u[r+1] = \Phi_V \wedge u[r+2] = \Phi_\chi \wedge \mathit{Before}(u[r+1], u[r+2]) \\
& \wedge u[r+3] = \mathit{current\ transaction\ id} \wedge u[r+4] = \infty \\
& \wedge \psi' \\
& \wedge \Gamma_\tau \\
& \wedge (\forall l)(1 \le l \le k.\ (\mathit{Before}(\Phi_\alpha, t_l[\mathrm{stop}]) \wedge \mathit{Before}(t_l[\mathrm{start}], \Phi_\beta))) \\
&)\}
\end{aligned}
$$

According to the predicates in the fourth line, a result should be:

Book#	Isbn	Shelf	Event time		Recording time	
			from	to	start	end
00065	0-999-99999-0	cb001	26/6/88	15/9/88	25/11/88	∞
00065	0-999-99999-0	cb003	15/9/88	∞	25/11/88	∞

We capture the first recording time (25/11/88) here. However, such a time attribute makes no sense for the observation of entity state! To present the exact semantics for the transaction, we need to deal with two recording times at the same time, like:

Book#	Isbn	Shelf	event time		recording time 3		recording time 1	
			from	to	start	to	start	to
00065	0-999-99999-0	cb001	26/6/88	15/9/88	1/10/88	∞	25/11/88	∞
00065	0-999-99999-0	cb001	15/9/88	∞	1/10/88	∞	25/11/88	∞

For the first tuple in this relation, it means that:

we made an observation on 1st October (and it was still valid before 25/11) about the entity state: Book# 00065 was on the shelf cb001 during the time 26/6 - 15/9; and such an observation was retrieved by the system on 25/11.

We have got two recording times for one relation! Now the system time is the first recording time (25/11/88), not the third one (1/10/88) as before. Maybe we would like to capture another recording time (20/10/88) as well, although it can be included in the interval (1/10/88 - ∞). They will cause the temporal schema to change after the transaction. Therefore, a temporal query language should support an *evolving temporal schema*, where the schema is allowed to change over recording time. How to capture the history of schema is a problem. To interpret the data correctly, one must create the appropriate data structure from the schema history information corresponding to the time that data was recorded. The implication is that several versions of the data structure may be handled even when a query requests only the current data.

Most temporal languages include no support for temporal schema evolution, even though non-temporal schema evolution (i.e., the evolution of user-defined schemas) [Lum et al. 1984]. If the user-defined schema does evolve there are clearly

implications for temporal databases that are distinct from those of temporal schema evolution. Thus, more study is needed for handling the evolution properly.

6.5 Integrity of Time Attributes

Because time is ordered, all of the tuples and attributes of the same entity in a temporal database are logically time-ordered. Data in temporal databases can be ordered in following three forms:

1) a temporal database is a sequence of historical relations indexed along the recording time, each historical relation indexed along event time;

2) we maybe would like data being ordered in entity, entity state, or observation of entity orders as well;

3) or we maybe only order the data in tuple level (indexed along the recording start time).

However, the tuples and attributes in traditional relational databases are totally unordered. The integrity of TDBMS is more difficult to maintain. It is necessary not only to maintain the integrity of entity (like in traditional databases), but also of entity state and of observation of entity. How to maintain them is an open question.

On the other hand, consider that the event time can be changed into assertion time with the changing history of enterprises, and think about the evolution of recording time, the complexity of maintaining the integrity among time attributes themselves can be imagined.

Finally, only one time attribute (recording time) is maintained by the system in the proposed approach. Therefore, the time order integrity of databases can only be guaranteed along this time dimension. Such a guarantee is not enough for TDBMS. While the event time attributes can be defined and specified by user (retrieved by the **when** clause and modified by the **valid** clause in TDB), the TDBMS cannot guarantee the integrity of event time values. For example, we maybe input wrong information into the BOOK-LOAN relation as follows: Book# 00023 was borrowed by Tony on 20/5/88. However, Book# 00023 was returned by Peter on 23/5/88.

Book#	Isbn	Name	Address	Date-due -back	Valid Time		Trans. Time	
					from	to	start	stop
00023	0-999-99999-0	Peter	Park Avn.	12/6/88	12/4/88	∞	12/4/88	23/5/88
00023	0-999-99999-0	Peter	Park Avn.	12/6/88	12/4/88	23/5/88	23/5/88	∞
00023	0-999-99999-0	Tony	Kelvin St.	20/7/88	20/5/88	∞	25/5/88	∞

According to the tuple calculus in Chapter 5, such a mistake cannot be detected automatically by the system. Further study is needed for the *time order integrity* of TDBMS.

6.6 Implementation

Most TDBMSs have not yet supported an implementable temporal query language, but some of them have a well-defined algebra [Snodgrass 1987, Bolour et al. 1982, McKenzie 1986]. An implementable language may be demonstrated formally through a semantics based on the algebra [Snodgrass 1987].

To implement, although the models may be based on some sort of traditional models (like relational model or hierarchic model), access methods such as sequential scan, hashing, and ISAM should be studied again. New storage structures are needed to obtain adequate performance [Snodgrass 1987]. To capture more temporal semantics, the concepts of entity, entity state, observation of entity, observation of entity state need more discussion and the syntax for them in the language should be researched. We also suffer from many semantics problems in the tuple calculus. Thus, to implement a good system, tuple calculus and the algebra for it need more extensions, e.g., the semantics for dealing with disjoint tuples.

Temporal database management systems show a lot of interesting attributes on capturing temporal semantics of information and maintaining temporal information. However, most of studies are at the conceptual levels. To implement such a system is still a long term study.

References

[Ahn 1986] Ahn, I., *Towards an implementation of database management systems, with temporal support*, Proceedings of the International Conference on Data Engineering, IEEE Press, New York, 1986

[Ahn & Snodgrass 1986] Ahn, I. and Snodgrass, R., *Performance Evaluation of a Temporal Database Management System*, Proceedings of ACM SIGMOD'86, SIGMOD Record, Vol.15 No.2, June 1986

[Allen 1983] Allen, J. F., *Maintaining Knowledge about Temporal Intervals,* Communications of the ACM, Vol.26, No.11, November 1983

[Anderson 1983] Anderson, T. L., *Modelling Events and Processes at the Conceptual Level,* Proceedings of the Second International Conference on Databases, Great Britain: Wiley Heyden Ltd., 1983

[Ariav 1986] Ariav, G., *A Temporally Oriented Data Model ,* ACM Transactions on Database Systems, Vol. 11, No. 4, December 1986

[Ariav & Morgan 1981] Ariav, G. and Morgan, H. L., *MDM: Handlling the time dimension in generalized DBMS,* Working Paper, Dept. of Decision Sciences, The Wharton School, University. of Pennsylvania, May 1981.

[Ben-Zvi 1982] Ben-Zvi, J., *The Time relational Model* , PhD. Dissertation, University of California, Los Angeles 1982

[Bolour et al. 1982] Bolour, A., Anderson, T. L., Deketser, L. J. and Wong, H. K. T.,*The role of time in information processing: A survey,* ACM SIGMOD Record, Spring 1982

[Bubenko 1977] Bubenko, J. A., Jr., *The Temporal Dimension in Information Modelling*, in "Architecture and Models in Data Base Management Systems", North-Holland Publish Co., 1977

[Bubenko 1980] Bubenko, J. A. Jr., *Information modelling in the context of system development,* Proceedings of IFIP Congress 80, 1980

[Clifford & Tansel 1985] Clifford, J. and Tansel, A. U., *On An Algebra For Historical Relational Databases: Two Views*, Proceedings of ACM SIGMOD International Conference on Management of Data. Ed.S. Navathe, May 1985

[Clifford & Warren 1983] Clifford, J. and Warren, D. S., *Formal Semantics for Time in Databases,* ACM Transactions on Database Systems, Vol.8, No.2, June 1983

[Codd 1979] Codd, E. F., *Extending the database relational model to capture more meaning,* ACM Transactions Database System, Vol.4, No.4, December 1979

[Copeland 1982] Copeland, G., *What If Mass Storage Were Free?*, IEEE Computer, Vol.15, No.7, July 1982

[Copeland & Maier 1984] Copeland, G. and Maier, D., *Making Smalltalk a Database System*, Proceedings of ACM SIGMOD International Conference on Management of Data, Ed. B. Yormark, Boston, June 1984

[Date 1982] Date, C. J., *A Formal Definition of the Relational Model* ACM SIGMOD Record, Vol.13, No.1, September 1982

[Date 1987] Date, C.J., *A Guide to the SQL Standard,* Addison-Wesley Publishing Company, 1987

[Findler & Chen 1971] Findler, N. and Chen, D., *On the problems of time retrieval, temporal relations, causality, and coexistence* , Proceedings of the International Joint Conference on Artificial Intelligence, Imperial College, September 1971

[Gadia 1988] Gadia, S. K., *A Homogeneous Relational Model and Query Languages for Temporal Databases*, ACM Transactions on Database Systems, Vol.13, No.4, December 1988

[Gadia & Yeung 1988] Gadia, S. K. and Yeung, C. S., *A Generalized Model for A Relational Temporal Dtabase*, Proceedings of ACM SIGMOD International Conference on Management of Data, 1988

[Galton et al. 1987] Galton, A. P. et al., *Temporal logics and their applications* Academic Press Limited, London 1987

[Garcia & Wiederhold 1982] Garcia-Molina, H. and Wiederhold, G., *Read-Only Transactions in a Distributed Database*, ACM Transactions on Database Systems, Vol.7, No.2, June 1982

[Held et al. 1975] Held, G.D., Stonebraker, M.R. and Wong, E., *INGRES - A relational data base system* , Proceedings of the 1975 National Computer Conference, Vol.44, AFIPS Press, 1975

[Jones & Mason 1980] Jones, S. and Mason, P. J., *Handling the time dimension in a database,* Proceedings of the International Conference on Data Base, Heyden, British Computer Society, July 1980

[Jones & Motro 1986] Jones, S. and Motro, A., *The Time Warp Mechanism for Database Concurrency Control*, Proceedings of the International Conference on Data Engineering, Los Angeles, CA, IEEE Computer Society Press, February, 1986

[Katz et al. 1986] Katz, R. H., Anwaruddin M. and Chang, E., *A Version Server for Computer-Aided Design Data*, 23rd Design Automation Conference Proceedings, ACM/IEEE, Las Vegas, NV, June 1986

[Klug 1982] Klug, A., *Equivalence of relational algebra and relational calculus query languages having aggregate functions,* J. ACM, Vol.29, No.3, July 1982

[Klopprogge 1983] Klopprogge, M. R., *Entity and Relationship Histories: A Concept for Describing and Managing Time Variant Information in Databases* , PhD. Dissertation, Universitat Karlsruhe, 1983

[Lum et al. 1984] Lum, V. et al., *Designing DBMS Support for the Temporal Dimension* , ACM SIGMOD'84, May 1984

[McDermott 1982] McDermott, D., *A temporal logic for reasoning about processes and plans,* Cognitive Science 6, 1982

[McKenzie 1986] McKenzie, E., *Bibliography: Temporal Databases,* ACM SIGMOD Record, Vol.15, No.4, December 1986

[McKenzie & Snodgrass 1987A] McKenzie, E. and Snodgrass, R., *Scheme Evolution and the Relational Algebra,* Technical Report TR87-003, Computer Science Department, University of North Carolina at Chapel Hill, March 1987

[McKenzie & Snodgrass 1987B] McKenzie, E. and Snodgrass, R., *Supporting Valid Time: An Historical Algebra and Evaluation,* Technical Report TR87-008, Computer Science Department, University of North Carolina at Chapel Hill, April 1987

[McKenzie & Snodgrass 1987C] McKenzie, E. and Snodgrass, R., *Extending the Relational Algebra to Support Transaction Time*, ACM SIGMOD Record, Vol.16, No.3, December 1987

[Oxborrow 1986] Oxborrow, E., *Databases and Database Systems: Concepts and Issues*, Chartwell-Bratt, 1987 printed

[Reed 1978] Reed, D., *Naming and Synchronization in a Decentralized Computer System* , PhD. Dissertation, M.I.T., September 1978

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

[Roth et al. 1988] Roth, M. A., Korth, H. F. and Silberschatz, A., *Extended Algebra and Calculus for Nested Relational Databases*, ACM Transactions on Database Systems, Vol.13, No.4, December 1988

[Sacerdoti 1977] Sacerdoti, E.D., *A Structure for Plans and Behavior,* Elsevier North-Holland, New York, 1977

[Schueler 1977] Scheler, B., *Update Reconsidered,* In "Architecture and Models in Data Base Management Systems", Ed. G. M. Nijssen-North Holland Publishing Co., 1977

[Segev & Shoshani 1987] Segev, A. and Shoshani, A., *Logical Modeling of Temporal Data* , ACM SIGMOD Record Vol.16, No.3, December 1987

[Sloman 1978] Sloman, A., *The Computer Revolution in Philosophy*, Hassocks, Harvester Press, 1978

[Snodgrass 1982] Snodgrass, R., *Monitoring Distributed Systems: A Relational Approach*, PhD. Dissertation, Computer Science Department, Carnegie-Mellon University, December 1982

[Snodgrass 1984] Snodgrass, R., *The Temporal Query Language TQuel*, Proceedings of the Third ACM SIGAct-SIGMOD symposium on Principles of Database Systems, April 1984

[Snodgrass 1986] Snodgrass, R., *Research Concerning Time in Databases Project Summaries*, SIGMOD Record, Vol.15, No.4, December 1986

[Snodgrass 1987] Snodgrass, R., *The Temporal Query Language TQuel*, ACM Transactions on Database Systems, Vol.12, No.2, June 1987

[Snodgrass & Ahn 1985] Snodgrass, R. and Ahn, I., *A Taxonomy of Time in Databases,* Proceedings of ACM SIGMOD International Conference on Management of Data, Ed.S. Navathe, May 1985

[Snodgrass & Ahn 1986] Snodgrass, R. and Ahn, I., *Temporal Databases,* IEEE Computer, September 1986

[Snodgrass & Gomez 1986] Snodgrass, R. and Gomez, S., *Aggregates in the temporal query language TQuel,* Technical Report TR86-009, Computer Science Dept., Univ. of North Carolina, Chapel Hill, March 1986

[Steedman 1977] Steedman, M. J., *Verb, time and modality*, Cognitive Science Vol.1(2), April 1977

[Ullman 1982] Ullman, J.D., *Principles of Database Systems*, 2nd ed. Computer Science Press, Rockville, Md., 1982

Appendix A. The syntax of TQuel [Snodgrass 1987]

TQuel augments five Quel statements: create, retrieve, append, delete, and replace.

<bool expression>	returns a value of type Boolean
<expression>	returns a value of type integer, floating point, or temporal
<attribute>	the name of an attribute
<relation>	a relation name
<string>	a string constant
<tuple variable>	the name of tuple variable
<attribute specs>	a list of the names and types of the user-specified attributes
ε	empty

<TQuel augmented>	::= <create stmt> | <retrieve stmt> | <append stmt> | <delete stmt> | <replace stmt>
<create stmt>	::= **create** <persistent> <history> <attribute specs>
<persistent>	::= ε | persistent
<history>	::= ε | interval | event
<retrieve stmt>	::= <retrieve head> <retrieve tail>
<retrieve head>	::= **retrieve** <into> <target list> <valid clause>
<retrieve tail>	::= <where clause> <when clause> <as-of clause>
<into>	::= ε | **unique** | <relation> | **into** <relation> | **to** <relation>
<target list>	::= ε | (<tuple variable> **.all**) | (<t-list>)
<t-list>	::= <t-elem> | <t-list>, <t-elem>
<t-elem>	::= <attribute> <is> <expression>
<is>	::= **is** | = | **by**
<append stmt>	::= **append** <to> <target list> <mod stmt tail>
<to>	::= <relation> | **to** <relation>
<delete stmt>	::= **delete** <tuple variable> <mod stmt tail>
<replace stmt>	::= **replace** <tuple variable> <target list> <mod stmt tail>
<mod stmt tail>	::= <valid clause> <where clause> <when clause>
<valid clause>	::= <valid> <from clause> <to clause> | <valid> <at clause>
<valid>	::= ε | **valid**
<from clause>	::= ε | **from** <e-expression>

<to clause> ::= ε | **to** <e-expression>
<at clause> ::= **at** <e-expression>
<where clause> ::= ε | **where** <bool expression>
<when clause> ::= ε | **when** <temporal pred>
<as-of clause> ::= ε | **as of** <e-expression> <through clause>
<through clause> ::= ε | **through** <e-expression>
<e-expression> ::= <event element>
| **begin of** <either-expression>
| **end of** <either-expression>
| (<e-expression>)
<i-expression> ::= <interval element>
| <either-expression> **overlap** <either-expression>
| <either-expression> **extend** <either-expression>
| (<>)
<either-expression> ::= <e-expression> | <i-expression>
<event element> ::= <tuple variable>
<interval element> ::= <tuple variable> | temporal constant>
<temporal constant> ::= <string>
<temporal pred> ::= <interval element>
| <event element>
| <either-expression> **precede** <either-expression>
| <either-expression> **overlap** <either-expression>
| <either-expression> **equal** <either-expression>
| <temporal pred> **and** <temporal pred>
| <temporal pred> **or** <temporal pred>
| <temporal pred>
| **not** <temporal pred>

Note that the **create** statement has not been discussed in the dissertation. It defines a new relation and provides a scheme for that relation. **Persistent**, **interval**, and **event** keywords are present with the **create** statement. If the **persistent** keyword is used, then the relation is either a rollback or a temporal relation. If the **interval** or **event** keyword is used, the relation is either a historical or temporal relation. If none of these keywords is used, the relation is a conventional snapshot relation.

Appendix B. TQuel Defaults [Snodgrass 1987]

The defaults assumed in the language will be important for the semantics of the language. The defaults for the additional clauses in TQuel will be discussed as follows. When one or more of these clauses are not provided by the user, it is assumed to take a defaults value. The user should be careful when only a few clauses are defaulted, because the defaulted clause(s) may be inappropriate.

The default of the **where** clause in TQuel is set as the default in Quel to "**where** true".

If only one tuple variable (say, *I*) is used, and it is associated with an interval relation, then the defaults of the retrieve statement are as follows:

valid from begin of *I* **to end of** *I*
when *I* **overlap** "*now*"
as of "*now*"

These defaults say that the result tuple is to start when the underlying tuple started and stop when the underlying tuple stopped and the query is to be executed on the current historical state.

When an event relation is associated with the one tuple variable (say, *E*) the default is

valid at *E*
when true
as of "*now*"

specifying simply that the result tuple was valid at the same instant the underlying tuple was valid.

When two or more tuple variables are used, and the tuple variables associated with interval relations involved in the query are $t_1, t_2, \ldots, t_k$, then the default temporal clauses are the following:

valid from begin of (t_1 **overlap** t_2 . . . **overlap** t_k) **to end of overlap**
(t_1 **overlap** t_2 . . . **overlap** t_k)
when (t_1 **overlap** t_2 . . . **overlap** t_k) **overlap** "now"
as of "now"

These clauses state that the underlying tuples must be consistent; that is, they are all valid for the entire interval over which the resulting tuple is valid. Tuple variables associated with event relations are ignored in this case.

For the **append** statement, the defaults are as follows:

valid from "now" **to** "forever"

when (t_1 **overlap** . . . **overlap** t_k) **overlap** "now"

This means that the tuples used to supply values for the new tuples to be appended should be currently valid, and that the new tuples should be considered to have become valid immediately.

For the **delete** statement, the defaults are as follows:

delete t_0

valid from "now" **to end of** t_0

when (t_0 **overlap** t_1 **overlap** . . . **overlap** t_k) **overlap** "now"

These defults imply that the deletion only applies to information valid now or in the future.

If t_0 was associated with an **event** relation, the default is as follows:

delete t_0

valid at t_0

when (t_0 **overlap**t_1 **overlap** . . . **overlap** t_k) **overlap** "now"

Note that in the original paper "t_0 **overlap**" did not appear; this is assumed to be a misprint.

For the replace statement, the defaults are as follows:

replace t_0

valid from "now" **to end of** t_0

when (t_0 **overlap** t_1 **overlap** . . . **overlap** t_k) **overlap** "now"

These defaults follow from the fact that a replace is roughly equivalent to a delete followed by an append. As discussed in Section 5.3.2.1, the default for the **when** clause in the **replace** statement should be that there is no **when** clause in it.

Appendix C. Semantics of TQuel [Snodgrass 1987]

The semantics of TQuel uses the snapshot relational database model as the underlying model of TDB by embedding the four-dimensional temporal relation in a two-dimensional snapshot relation. In this way the semantics can be expressed in a traditional tuple calculus formalism.

C.1 Quel semantics

The tuple calculus statement for the skeletal Quel statement

range of t_1 **is** R_1

. . . .

range of t_k **is** R_k

retrieve $(t_{i_1}.D_{j_1}. . . ,t_{i_r}.D_{j_r})$

where ψ

is

$$\{u^{(r)} | (\exists t_1) \ldots (\exists t_k)(R_1(t_1) \wedge \ldots \wedge R_k(t_k) \wedge u[1] = t_{i_1}[j_1] \wedge \ldots \wedge u[r] = t_{i_r}[j_r] \wedge \psi')\}$$

Refer to the section 4.2.1.2. No complete formal semantics of Quel has been specified. Ullman has defined a tuple relational calculus semantics for Quel statements without aggregates [Ullman 1982], and Klug has treated aggregates in the more general case [Klug 1982].

C.2 TQuel retrieve statement's semantics

The tuple calculus statement for a TQuel retrieve statement is very similar to that of a Quel retrieve statement; additional components corresponding to the **valid**, **when**, and **as-of** clauses are also present. Although the expressions appearing in all three clauses are similar syntactically, having their origins in path expressions, their semantics are quite different.

A formal semantics for the TQuel retrieve statement can be specified as follows. Let Φ_e be the function corresponding to the e-expression e. Let Γ_τ be the predicate corresponding to the temporal predicate τ. Φ_e and Γ_τ will contain only the functions *First* and *Last* and the predicates *Before* , $\wedge$, $\vee$, $\neg$; the rest of the functions, and Φ_α (where α appears in an **as-of** clause), can be entirely evaluated at "compile time." Given the query

range of t_1 **is** R_1

. . . .

range of t_k **is** R_k

retrieve $(t_{i_1}.D_{j_1}. . . ,t_{i_r}.D_{j_r})$

valid from ν **to** χ

where ψ

when τ

as of α **through** β

the tuple calculus statement has the following form:

$$\{u^{(r+4)} | (\exists t_1) \ldots (\exists t_k)(R_1(t_1) \wedge \ldots \wedge R_k(t_k)$$
$$\wedge u[1] = t_{i_1}[j_1] \wedge \ldots \wedge u[r] = t_{i_r}[j_r]$$
$$\wedge u[r+1] = \Phi_\nu \wedge u[r+2] = \Phi_\chi \wedge Before(u[r+1], u[r+2])$$
$$\wedge u[r+3] = current\ transaction\ id \wedge u[r+4] = \infty$$
$$\wedge \psi'$$
$$\wedge \Gamma_\tau$$
$$\wedge (\forall l)(1 \le l \le k.(Before(\Phi_\alpha, t_l[stop]) \wedge Before(t_l[start], \Phi_\beta)))$$
$$)\}$$

The first line states that each tuple variable ranges over the correct relation. The resulting tuple consists of r explicit attributes and four implicit attributes (*from, to, start, and stop*). The second line states the origin of the values in the explicit attributes of the derived relation. The third line originates in the **valid** clause and specifies the values of the *from* and *to* valid times. Note that these times must obey the specified ordering, i.e., *from* is before *to*. The fourth line specifies the values of the *start* and *stop* transaction times. "*current transaction id* " is replaced with an integer corresponding to the current transaction time (time granuarity is assumed to be calandar date in this dissertation). The next line (ψ') originates in the **where** clause and is from the Quel semantics. The sixth line (Γ_τ) is the predicate from the **when** clause. The last two lines originate in the **as-of** clause and states that the tuple associated with each tuple variable must have a transaction interval that overlaps the interval specified in the **as-of** clause (Φ_α and Φ_β will be constant time values).

C.3 The semantics of TQuel modification statements

C.3.1 Append

The semantics for the skeletal TQuel **append** statement is:

range of t_1 **is** R_1

. . . .

range of t_k **is** R_k
append to R $(ti_1.Dj_1. \ldots ,ti_r.Dj_r)$
valid from ν **to** χ
where ψ
when τ

then the tuple calculus statement for interval append is redefined as the following form:

$$
\begin{aligned}
R' = R \cup \{ u^{(r+4)} \mid & (\exists t_1) \ldots (\exists t_k)(R_1(t_1) \wedge \ldots \wedge R_k(t_k) \\
& \wedge (\forall l)\ (1 \le l \le r.\ u[l] = t_{i_l}[j_l]) \\
& \wedge u[r+3] = \textit{current transaction id} \wedge u[r+4] = \infty \\
& \wedge \psi' \\
& \wedge \Gamma_\tau \\
& \wedge (\forall l)(1 \le l \le k.t_l\ [\text{stop}] = \infty) \\
& \wedge ((\exists s)(R(s) \wedge (\forall l)\ (1 \le l \le r.\ s[l] = u[l]) \\
& \qquad \wedge (C_1{}^a \vee C_2{}^a \vee C_3{}^a \vee C_4{}^a)) \\
& \vee ((\exists s)(\neg R(s) \wedge (\forall l)\ (1 \le l \le r.\ s[l] = u[l]) \\
& \qquad \wedge u[r+1] = \Phi_\nu \wedge u[r+2] = \Phi_\chi))) \\
&)\}
\end{aligned}
$$

where,

$C_1{}^a = (\text{Before}(s[r+1], \Phi_\nu) \wedge \text{Before}(\Phi_\nu, s[r+1]) \wedge \text{Before}(s[r+2], \Phi_\nu)$
$\wedge\ u[r+1] = s[r+2] \wedge u[r+2] = \Phi_\chi)$

$C_2{}^a = ((\text{Before}(\Phi_\nu, s[r+1]) \wedge \text{Before}(s[r+2], \Phi_\chi))$
$\wedge ((u[r+1] = s[r+2] \wedge u[r+2] = \Phi_\chi) \vee (u[r+1] = \Phi_\nu \wedge u[r+2] = s[r+1]))$

$C_3{}^a = (\text{Before}(\Phi_\nu, s[r+1]) \wedge \text{Before}(s[r+1], \Phi_\chi)$
$\wedge\ \text{Before}(\Phi_\chi, s[r+2]) \wedge u[r+1] = \Phi_\nu \wedge u[r+2] = s[r+1])$

$C_4{}^a = (\text{Before}(s[r+1], \Phi_\nu) \wedge \text{Before}(\Phi_\chi, s[r+2]) \wedge \text{False})$

The four clauses $C_1{}^a$, $C_2{}^a$, $C_3{}^a$ and $C_4{}^a$ handle the various overlap situations between the tuples to be added and the tuples identical in the explicit attributes that already exist during this valid interval. The last two lines of the expression for the derived relation R' state that the valid times are as specified in the **valid** clause if no such tuples exist during this valid interval.

C.3.2 Delete

The TQuel **delete** statement of intervals is:

range of t_1 **is** R_1

. . . .

range of t_k **is** R_k

range of s **is** R

delete s

valid from ν **to** χ

where ψ

when τ

then the tuple calculus statement is:

$$
\begin{aligned}
R' = \{u^{(r+4)} \mid\ & (\exists t_1) \ldots (\exists t_k)(\exists s)(R_1(t_1) \wedge \ldots \wedge R_k(t_k) \\
& \wedge (\forall l)(1 \le l \le k . t_l\ [\text{stop}] = \infty) \\
& \wedge (\forall l)\ (1 \le l \le r .\ u[l] = s[l]) \wedge u[r+1] = s[r+1] \\
& \wedge u[r+2] = s[r+2] \wedge u[r+3] = s[r+3] \\
& \wedge ((\neg \text{Affected} \wedge u[r+4] = s[r+4]) \\
& \vee (\text{Affected} \wedge u[r+4] = \textit{current transaction id}))\} \\
\cup \{u^{(r+4)} \mid\ & (\exists t_1) \ldots (\exists t_k)(\exists s)(R_1(t_1) \wedge \ldots \wedge R_k(t_k) \\
& \wedge (\forall l)(1 \le l \le k . t_l\ [\text{stop}] = \infty) \\
& \wedge (\forall l)\ (1 \le l \le r .\ u[l] = s[l]) \wedge \text{Affected} \\
& \wedge (C_1^{\,d} \vee C_2^{\,d} \vee C_3^{\,d} \vee C_4^{\,d}) \\
& \wedge u[r+3] = \textit{current transaction id} \wedge u[r+4] = \infty \\
&)\}
\end{aligned}
$$

where,

$$
\begin{aligned}
\text{Affected} = (R(s) \wedge \psi' \wedge \Gamma_\tau \wedge s[r+4] = \infty & \\
\wedge (\text{Before}(s[r+1], \Phi_\chi) \wedge \text{Before}(\Phi_\nu, s[r+2])) & \\
C_1^{\,d} = (\text{Before}(s[r+1], \Phi_\nu) \wedge \text{Before}(\Phi_\nu, s[r+2]) \wedge \text{Before}(s[r+2], \Phi_\chi) & \\
\wedge u[r+1] = s[r+1] \wedge u[r+2] = \Phi_\nu) & \\
C_2^{\,d} = (\text{Before}(\Phi_\nu, s[r+1]) \wedge (\text{Before}(s[r+2], \Phi_\chi) \wedge \text{False}) & \\
C_3^{\,d} = (\text{Before}(\Phi_\nu, s[r+1]) \wedge \text{Before}(s[r+2], \Phi_\chi) \wedge \text{Before}(\Phi_\chi, s[r+2]) & \\
\wedge u[r+1] = \Phi_\chi \wedge u[r+2] = s[r+2]) & \\
C_4^{\,d} = ((\text{Before}(s[r+1], \Phi_\nu) \wedge \text{Before}(\Phi_\chi, s[r+2])) & \\
\wedge ((u[r+1] = \Phi_\chi \wedge u[r+2] = s[r+2]) \vee (u[r+1] = s[r+1] & \\
\wedge u[r+2] = \Phi_\nu)) &
\end{aligned}
$$

There are two sets in the tuple calculus statements of **delete** operation. One contains all tuples in past historical relations of R and all tuples in the current historical relation of R that are not **Affected**, that is, that do not satisfy the predicate in the **where** or **when** clauses or whose valid intervals do not overlap with the specified valid interval. Another set deals with the existing tuples that only partially should be deleted. Those portions that should not have been deleted are added back in the second set. The clauses $C_1^{\,d}$, $C_2^{\,d}$, $C_3^{\,d}$, and $C_4^{\,d}$ calculate the valid times of those tuples.

3.3 Replace

The TQuel **replace** statement of intervals is:

range of t_1 **is** R_1

. . . .

range of t_k **is** R_k

range of s **is** R

replace s($ti_1.D_{j1}$. . . ,$ti_r.D_{jr}$)

valid from ν **to** χ

where ψ

when τ

then the tuple calculus statement for interval replacing has the following form:

$$
\begin{aligned}
R' = \{u^{(r+4)} \mid\ & (\exists t_1) \ldots (\exists t_k)(\exists s)(R_1(t_1) \wedge \ldots \wedge R_k(t_k) \\
& \wedge (\forall l)(1 \le l \le k . t_l\ [\text{stop}] = \infty) \\
& \wedge (\forall l)\ (1 \le l \le r .\ u\ [l\ ] = s\ [l\ ])\ \wedge u\ [r+1\ ] = s\ [r+1\ ] \\
& \qquad \wedge u\ [r+2\ ] = s\ [r+2\ ]\ \wedge u\ [r+3\ ] = s\ [r+3\ ] \\
& \wedge ((\neg \text{Affected} \wedge u\ [r+4\ ] = s\ [r+4\ ]) \\
& \quad \wedge (\text{Affected} \wedge u\ [r+4\ ] = \textit{current transaction id}\ ))) \} \\
\cup \{u^{(r+4)} \mid\ & (\exists t_1) \ldots (\exists t_k)(\exists s)(R_1(t_1) \wedge \ldots \wedge R_k(t_k) \\
& \wedge (\forall l)(1 \le l \le k . t_l\ [\text{stop}] = \infty) \\
& \wedge (\forall l)\ (1 \le l \le r .\ u\ [l\ ] = s\ [l\ ]) \wedge \text{Affected} \\
& \wedge (C_1{}^d \vee C_2{}^d \vee C_3{}^d \vee C_4{}^d) \\
& \wedge u\ [r+3\ ] = \textit{current transaction id}\ \wedge u\ [r+4\ ] = \infty \\
&)\} \\
\cup \{u^{(r+4)} \mid\ & (\exists t_1) \ldots (\exists t_k)(R_1(t_1) \wedge \ldots \wedge R_k(t_k) \\
& \wedge (\forall l)\ (1 \le l \le r .\ u\ [l\ ] = t_{il}\ [j_l\ ]) \\
& \wedge u\ [r+3] = \textit{current transaction id}\ \wedge u\ [r+4] = \infty \\
& \wedge \psi' \\
& \wedge \Gamma_\tau \\
& \wedge (\forall l)(1 \le l \le k . t_l\ [\text{stop}] = \infty) \\
& \wedge ((\exists s)(R(s) \wedge (\forall l)\ (1 \le l \le r .\ s\ [l\ ] = u\ [l\ ]) \\
& \qquad \wedge (C_1{}^a \vee C_2{}^a \vee C_3{}^a \vee C_4{}^a)) \\
& \quad \vee ((\exists s)(\neg R(s) \wedge (\forall l)\ (1 \le l \le r .\ s\ [l\ ] = u\ [l\ ]) \\
& \qquad \wedge u\ [r+1] = \Phi_\nu \wedge u\ [r+2] = \Phi_\chi))) \\
&)\}
\end{aligned}
$$

Here, the **replace** statements are exactly equal to the **delete** statements combining with the **append** statements. Three sets are processed. The first two of them are like

the statements in **delete** operation, the third is almost like those in the **append** statements. The third set appends the tuples with those portion which have not existed in the valid interval; and it appends the new tuples which *have not existed* during the entire valid time as the predicates stated in the last two lines. However, the replace operation cannot replace any non-existent tuple. That is, it is *not necessary* to append the tuples which have not existed during the valid interval to the database. Therefore, the predicates in the last two lines are redundent (c.f., Chapter 5).

Appendix D. The Syntax of TOSQL [Ariav 1986]

<query>::= <b-query> <obj-spec> <time-spec> <time-qualif>

<b-query>::= SELECT <att-spec> FROM <cube-name>

<att-spec>::= * | <att_1>,..., <att_n>

<cube-name::= a name of a properly defined database cube

<obj-spec>::= ALL-OBJECTS | WHERE <selection-expression>
<prevalence-mode>

<selection-expression> relates attribute-references and literals through comparison operators (e.g., =, >, <), or Boolean operators (AND, OR, and NOT). Parentheses enforce desired order of evalution.

<prevalence-mode>::= EVERYWHEN | SOMEWHEN

<time-spec>::= <time-period> <time-dimension>

<time-period>::= AT <time-point>
| WHILE <selection-expression> <temp-boundaries>
| DURING (<t> - <t>)
| BEFORE <t>
| AFTER <t>

<time-dimension>::= ALONG RT | ALONG <tsa>

<time-point>::= PRESENT | <t>

<temp-boundaries>::= DURING (-∞ - +∞) | DURING (<t> - <t>)

<t>::= time value, in Chronons

<time-qualif>::= AS-OF <time-point> <time-dimension>
