ASSESSING THE TEMPORAL DIFFERENTIATION OF ATTRIBUTES AS AN IMPLEMENTATION STRATEGY FOR A TEMPORALLY ORIENTED RELATIONAL DBMS

by

Joseph Shiftan

Information Systems Area
Graduate School of Business Administration
New York University
90 Trinity Place
New York, N.Y. 10006

August 1986

Center for Research on Information Systems
Information Systems Area
Graduate School of Business Administration
New York University

Working Paper Series

CRIS #159
GBA #87-62

Acknowledgements

I would like to thank my dissertation committee, Professors Gadi Ariav, Jim Clifford, Norman White, Zvi Kedem and Al Croker for their comments and suggestions in the development and completion of this dissertation. My two advisors, Professors Gadi Ariav and Jim Clifford dedicated a lot of time and effort in numerous discussions that we had in the process of conducting this research, and in reviewing previous versions of this dissertation. I am very grateful to them, and I hope to continue the productive cooperation with them in the future. Finally, I would like to thank my family for their support during the period of my doctoral studies.

Joseph Shiftan
Jospeh Shiftan

Abstract: Assessing the Temporal Differentiation of Attributes as an Implementation Strategy for Temporally Oriented Relational Databases

Joseph Shiftan - New York University, GBA, CAIS
(Advisors: Professors Gad Ariav and Jim Clifford)

Temporally Oriented Databases (TODBs) are database systems in which both historical and current data are accessed and treated with full symmetry. The growing interest in such systems is manifested recently in a number of research efforts focusing on a wide set of issues, ranging from the study of abstract conceptual models to the practical implementation of working systems.

Attempts to implement TODBs have so far been at best preliminary, characterized by an ad hoc flavor, or have had a very limited scope. This dissertation research is an attempt to design a general purpose relational Temporally-Oriented Database Management System (TDMS), and examine the feasibility of its implementation along current theoretical concepts. The users view data in a TDMS as a temporally oriented, three dimensional cube; this is, in fact, implemented as a two layered data structure. The implementation model interrelates the external user view with an underlying functional view of the data, and specifies on the translation between these layers.

The major principle in the implementation is the differentiation of attributes according to their temporal variation. This research uses this concept as an implementation strategy of TDMSs, and assesses this approach for dealing with the following primary questions: efficient ways to store and retrieve data, the integrity constraints needed to maintain the database consistency and the definitions and implementations of temporal operations in such systems.

Further validation of the model was achieved through the development of a TDMS prototype. The prototype was developed using INGRES commands embedded in PASCAL programs on VAX/VMS, and provides a test bed for further studies of temporally oriented information systems.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction and Overview

1.1. The Motivation of this Research

There is intensifying research interest in expanding the capabilities of Database Management Systems (DBMSs). The current state of general purpose DBMSs is the result of many well documented research efforts, for example [Codd 70], [Stonebraker 76], [Chen 76] and [Ullman 80]. Current DBMSs, however, typically assume that the user is interested only in the present (most recent) data, and do not provide him with adequate tools to deal with data at other points in time. The user can, of course, explicitly define time as a data item in his/her database, and use it to "tag" time-varying data items (e.g., salary). However, this only allows for a limited, schema-defined, number of time-points for which values are stored. In addition, the user himself is burdened with defining and implementing the necessary operations to capture the full meaning of time in storing and retrieving data.

The absence of a DBMS that handles time properly is being challenged by the growing need to have the same quality of access to both historical and present information, and to treat them uniformly. This need has been prompted by a number of developments. Prominent among them is the concept and the growing usage of Decision Support Systems (DSSs) [Morgan 81], [Ginzberg 82].

The capacity to deal adequately with time and historical information is a core requirement in many systems [Ariav 83a]. Historical information is also an essential component in business planning in general, and especially in "Retrospective Analysis" [Ackoff 81]. The time has come for developing a new type of database, a Temporally-Oriented Database (TODB), to meet the needs for maintenance of historical data. In handling such a database, both historical and current data should be treated uniformly and accessed with the same ease and with fully symmetric functionality. Recognizing the need for systems that provide these capabilities has already motivated many research efforts, summarized in [Bolour 82]. This survey contains about forty references to the subject, addressing largely four topics:

* Conceptual data-modeling (12 references), e.g., an extension to the relational model to incorporate a built-in semantics for time [Bubenko 77], [Clifford 82a], [Clifford 83a], [Ariav 83a].

* Design and implementation of historical databases (10 references), e.g., the organization of write-once, historical database [Ariav 81], [Ben-Zvi 82] and implementation of temporally-oriented medical databases [Wiederhold 75], [Fries 72].

* "Dynamic databases" (6 references), e.g., the modeling of transition rules and temporal inferences from these rules [Mays 81], [Findler 71].

* AI related research (12 references), e.g., the temporal understanding of time-oriented data [Kahn 75].

In a parallel development, constraints that traditionally have limited the practicality and feasibility of TODBs, e.g., the huge amount of storage needed for maintaining a "complete history", have recently been loosened in the wake of new hardware developments such as optical storage technologies, e.g., [Copeland 82], [Chi 82].

A panel held in summer 1983 brought together many researchers in the field, to discuss their work and identify promising research areas [Ariav 83b]. This panel pointed out that one of the areas requiring further research effort was the implementation of TODBs. The designation of this effort as a high priority item on the research agenda of TODBs is also included in other sources, e.g., [Ariav 83a] , [Ben-Zvi 82].

Since the [Bolour 82] survey, more research has been reported, including some efforts that concentrate mainly on the design and implementation of TODBs, and on the query languages needed to support them, e,g,, [Lum 84], [Snodgrass 84], [Snodgrass 85], [Ariav 85] and [Clifford 85a].

The major part of the theoretical efforts in this area deal with a temporal extension to the relational data model. Consequently, our research attempts to study the design and implementation of a general purpose relational Temporally-Oriented Database Management System (TDMS), capable of managing varied information, completely independent of the nature of the attributes and the data structures defined to contain them. This TDMS retains some of the major properties of the relational model, within which this implementation research is carried out.

In this dissertation, the terms TODB and TDMS are mentioned frequently. It should be clear, that the first refers to the database itself, while the second is the system that handles this database. Therefore, whenever dealing with the content of the database, the term TODB is used, and when describing the properties of systems handling such databases, the term TDMS is used.

1.2. The Research Goals

In most existing information systems, time aspects are usually either ignored, treated only implicitly, or factored out [Tsichritzis 82]. Most DBMSs do not treat present and past related questions symmetrically, but typically differentiate between them in terms of data accessibility. It is important to emphasize that this approach prevails not because of the scarcity of temporal references in common data, but rather in spite of their abundant availability. These current practices clearly simplify the task of data management, but result in reduced functionality and a less "correct" or "faithful" representation of reality.

This research focuses on the assessment of TDMS with respect to practicality, and ultimately validates this assessment through the conceptualization, design and implementation of a general purpose system that preserves the inherent dynamics of its content, deals with it explicitly, and makes the time dimension of the data accessible to its users. The TDMS has been developed within the context of the relational database model [Codd 70], [Codd 74], [Codd 79], [Maier 82], [Merrett 84], utilizing theoretical concepts that have already been suggested (e.g., [Ariav 83a], [Ariav 83c], [Clifford 82a], [Clifford 82b], [Clifford 83a], [Clifford 83b], [Ben-Zvi 82]). The basic component of these conceptual views of time-varying data is viewing the data items as organized in a cube, creating a three-dimensional, temporal extension of the relational model. Two of the dimensions in this cube are identical to those of the regular relational model (objects and attributes), and the third dimension is time. We call this cube of information a Temporally Oriented Relation (TOR).

The storage of TORs, which seems at first to be a simple one, is actually quite complicated. A cube of data, as viewed by the user, conceptually contains a full description of the relation for all time-points within some time period. Therefore, a direct implementation would, if possible at all, be highly redundant and impractical, and the basic question that should be addressed is how to implement these TORs efficiently, both for storage and retrieval [Clifford 83a]. Furthermore, a database will probably contain a mixture of static information (NAME, BIRTHDATE, etc.) and time-varying items (SALARY, POSITION, etc.), and a TDMS must handle both intelligently.

Finding an efficient way to store and retrieve the data is, of course, not enough. As we know, the real power of the relational model is in its operations that allow the user of a relational database either to select portions of the information included in a specific relation, or to join two relations and create a new relation containing their combined information.

Following the regular relational model, our TDMS should also have the capability to answer queries by executing operations on the TORs. Manipulations of TORs may be specified with or without explicit temporal components, and they should be expressed in terms of the user's cubic view, and then translated into efficient manipulations of the implemented data structures. These operations should be natural extensions to the basic regular relational algebra operations, SELECT, PROJECT and JOIN, defined in the relational model [Maier 82], [Ullman 80]. The collection of these new operations are referred to as Temporal Relational Algebra Operations. Although they are extensions of the regular operations, the complexity

introduced by adding the temporal dimension to the relational model puts slightly different interpretations on some of the temporal operations, and makes their implementation depend upon the characteristics of the TORs involved. However, like the regular relational model, these operations are the major component that makes our system powerful and responsive to the typical user's needs.

As mentioned above, there have been already attempts to deal with the theoretical concept of TDMSs, emphasizing the mechanism of uniform access to current and historical data (e.g., [Ariav 83a] and [Clifford 83b]). In this research, we carry the task a bit further, and use the theoretical views developed in these works as a basis for a comprehensive and rigorous framework for the actual design and implementation of such systems. The proposed framework emerges out of conceptual considerations, and, therefore, provides sound guidelines for implementation and a meaningful basis for evaluation.

1.3. Temporal Differentiation of Attributes

The major concept developed and examined in this research is the temporal differentiation of attributes, and its use as a basis for an implementation-level data model of temporally oriented relational databases. According to this concept, the time stamps in our data structures are associated with the various attributes in the TORs, rather than with entire tuples. This research investigates the implications of this approach for the implementation of relational TODBs.

Temporal differentiation of attributes rests on the following

observation. In regular relations, there is only one type of atomic value associated with the various attributes, and key attributes are distinguished from non-key attributes. TORs, however, are more complicated, as not all attributes change in the same way along time. There are, of course, key attributes, but the rest of the attributes are divided into at least two different kinds: constant attributes (CAs) (e.g., NAME, PLACE_OF_BIRTH), which are time invariant, and time-varying attributes (VAs) (e.g., SALARY, ADDRESS) [Clifford 85a]. In addition, one could further distinguish between interpolatable and non-interpolatable time-varying attributes. For example, if the salary of an employee is known at two time-points, and there is no salary value for any time point between them, it can be usually concluded that his salary in every intermediate time-point is the same as in the earlier time-point. This is likely to be the appropriate interpolation operation for this kind of data. Now, suppose that a database records car accidents with the time stamps in which they occur. If someone has two accidents in two different days, then it does not imply anything about the time between these two dates. This is a case of a non-interpolatable time-varying attribute. Similarly, knowing the sales volume on Monday and on Wednesday implies nothing about Tuesday. Basically, time varying data are either states (like position), or events (like car accidents). Each of them should be handled somewhat differently, but both of them are completely different from constant attributes.

The basic distinction between constant attributes, which do not change at all along time, and the various types of time-varying attributes, can be

powerfully exploited, to provide solutions to the problems involved in implementing TODBs. The temporal differentiation of attributes is a storage strategy that recognizes explicitly the fact that various attributes vary differently over time. Specifically, attributes are stored independently of each other, clustered along their pattern of change over time. If time stamps are associated with the whole tuple, then a change in one of its attributes implies the creation of a new tuple containing the new value of this attribute and the appropriate time stamp. No other attributes in this tuple may have changed, and yet they have to be copied to the new tuple, together with the one that has, in fact, changed. This situation introduces a lot of redundancy into the database.

In designing the TDMS, we use our basic approach - the temporal differentiation of attributes - and focus on a comprehensive assessment of this concept as an implementation strategy of Temporally Oriented Relational Databases.

The most general formulation of the key questions in our research is, therefore, as follows:

* What is the impact of the differences between attribute types in TODBs on the design and the implementation of TDMSs?

* How can these differences be exploited in the design of the data structures in the system, and how will these data structures affect all the other components of the TDMS?

1.4. Previous Research Efforts in Designing TODBs

Before presenting the research goals and methodologies, we turn to highlight earlier efforts to design and implement Temporally-Oriented Databases. We have selected a representative sample that demonstrates the current status of efforts in the domain. In particular, we distinguish among four categories of efforts:

1. Theoretical basis developed, but no design is explicated: [Bubenko 77], [Clifford 82b].
2. Theoretical basis explicated, design outlined, but no actual implementation attempted: [Ben-Zvi 82], [Ariav 83a].
3. Theory-based implementation of limited aspects of TODB [Snodgrass 84], [Lum 84].
4. Ad hoc implementation, without an explicit theoretical basis: [Wiederhold 75], [Ariav 81].

The underlying premise of the expanding body of research in this field is the recognition that time is not merely another attribute, or another data item tagged along with each tuple, but a dimension that requires new and different conceptual tools and design techniques.

1.4.1. Theoretical Studies Without Any Implementation

[Bubenko 77] illustrates the problems in handling temporal data through an example of an inventory management system. The system keeps track of available quantities-on-hand, and the events affecting this information, viz., shipments and deliveries. Bubenko argues that time should be introduced through recording the quantities-on-hand at times of a change. This method guarantees that the database includes a finite number of time-point references.

Handling time requires special care, he argues, and cannot be treated as an integral part of any traditional data model. For instance, the relation QUANTITY-ON-HAND(article,quantity,time) is not a relation in the conventional sense, since if we define its meaning to be "quantity at time t" where t is a variable, its extension includes an infinite number of tuples. However, assuming that there is an underlying finite representation of the database, time views could be defined using the tools of relational algebra or relational calculus.

The latter point is one of the major issues dealt with in [Clifford 82a]. This work provides the theoretical background for the incorporation of a temporal component into the relational model. It shows how the semantics of such an extended relational model can be mirrored by the semantics of formal logic. In particular, it identified three major principles that must be addressed in any practical implementation of an historical database system. The first of these was the notion of a "completed relation," which motivated the intuitive concept of the database as a collection of three-dimensional "cubes" of facts. The Comprehension Principle, a three-dimensional version of the "closed-world" assumption [Reiter 78], stated that the database is assumed to contain complete information about the objects throughout the time period which it is capturing. Finally, the need for explicit "Continuity Assumptions" was identified. According to this assumption, the cube is dense and includes values for all time points in the temporal dimension. This paper also defines functions that are associated with time-varying attributes in the model, and a mapping from a partial specification of their value (for example,

from a sampling at a small number of points) to a complete specification of their value over some time interval. While implementation was not directly addressed in this work, these principles are in fact very important in any implementation effort.

1.4.2. Tentative Designs

Some research efforts tried to form a theoretical basis for a design of a TDMS. For instance, [Ben-Zvi 82] introduced the Time Relational Model as a basis for a new architecture which incorporates comprehensive time processing capabilities into the relational model. In it, a traditional relation is extended into a time-relation which contains the tuples' history, i.e., all the values that each tuple has acquired over time.

A time view operator, TV, is then defined as the operation that extracts from a time-relation a regular ("flat") relation which corresponds to the state of affairs at a specified point in time (as seen from any specified, possibly different, point in time). Every operation on data is preceded by the TV operator, and therefore retains its standard relational meaning. The model proposes a unified view of present and past data, and thus maintains time-independence and time transparency, i.e., the user may operate on this model without having a special concern for time.

Although no actual implementation is reported, some aspects of TODB's design are discussed. For instance, a distinction is drawn between the "interpretive" and the "built-in" approaches. The former adds a time relational front-end interface to an existing DBMS (the approach we have

chosen), while in the latter a time relational DBMS is designed and built from scratch (probably trading off design effort for execution efficiency). The proposed database is arranged such that "current" tuples are stored physically together, and all past versions of the tuples are stored in a second storage area. All tuples of the same kind are chained together in descending order of their time stamps.

The proposed structure includes rather elaborate indexing, to allow efficient retrieval of current and past tuples. Interestingly, the proposal includes a facility to support future data, by automatically replacing current data with future data at the appropriate time. This is achieved by maintaining a specialized chain, called the tuple-future chain.

A clear limitation of the time-relational model is that it does not include any relational algebra operators that act directly upon the time-relations, and therefore none of the problems associated with such operators are actually addressed. This issue was nevertheless the major concern in [Ariav 83a], where the data model developed in it includes the data-cube as a basic data construct, and a set of operations and constraints on such constructs are outlined. The dimensions of the cube are objects, attributes, and time, each of which could be directly manipulated by the user. The temporal aspect of data is not factored out when the user accesses it. This ubiquitous cubic view of temporally-oriented data serves as the external (users') view of historical data in this dissertation research.

[Ariav 83a] used the model as the basis for the design of a TODB, and in

particular for the design of an extension of SQL to incorporate in it temporal elements in a way that does not penalize users who are not interested in access to the historical data. The query language is then expressed in a set of navigational operations, and ultimately translated into a DBTG schema. The same model is then used in the design of an integrated graphic user-interface that introduces the time dimension into query responses.

In the above system, not only the information is time stamped, but also the schema. The system supports multiple, time-ordered (internal) schemata, through which programs will gain access to the portion of data that has been recorded while the corresponding schema "prevailed".

Both [Ben-Zvi 82] and [Ariav 83a] developed some limited implementation design, but did not validate their architectural proposals through a corresponding detailed design or a prototyping effort.

1.4.3. Limited Implementations

[Snodgrass 84] has developed and implemented TQUEL for querying temporal databases. TQUEL is a superset of QUEL, the query language in the INGRES relational database management system. A tuple relational calculus semantics is provided for the TQUEL RETRIEVE statement. Two additional temporal constructs have been defined in TQUEL: the WHEN and VALID clauses, direct semantic analogies to QUEL's WHERE clause and target list.

The time stamps used in the data model underlying [Snodgrass 84] are associated with the whole tuple, an approach that has prevailed so far.

Incidently, the problem of the storage redundancy does not show up in the examples illustrating TQUEL in Snodgrass's paper, due to their simplicity (each of the relations in these examples contains only one time-varying attribute in addition to the key-attribute, the NAME, a temporally constant attribute). However, in a more realistic situation one would expect to have more time-varying attributes, like SALARY, MARITAL STATUS, DEPARTMENT, ADDRESS, etc., which should be assumed to vary in different time-points. Such a situation cannot fit elegantly into TQUEL. These fundamental problems limit to a large extent the practicality of TQUEL.

TQUEL distinguishes between event relations and interval relations. Interval relations have two time stamps in each of their tuples, called "start" and "stop". The absence of an open-ended category forces every continuous state to end at an explicitly specified time-point, which is rather cumbersome. A common situation is that a state has started at some time-point, but has not yet ended as of today. Also, there is a lot of redundant data when the "stop" of one tuple is automatically the "start" of the subsequent.

[Lum 84] proposes a method, which has been validated through implementation, to handle historical databases. Its basic design supports the usual view of database functions, and, in addition, the general time domain, so that both current and historical data can be accessed in the same way. Like [Snodgrass 84], [Ariav 83a], [Clifford 82a], [Ben-Zvi 82] and others, [Lum 84] extends the relational database model, and they view the temporal data as a three-dimensional structure.

The implementation approach used by [Lum 84] is the common one, associating time stamps with the whole tuple. In each relation there is a table containing only the current tuples, as in regular DBMSs, but in addition a transparent time stamp is added to each tuple. Included also with each tuple is a pointer to the first history tuple, if any. All history information belonging to one tuple is chained in reverse time order, with the beginning of the chain always in the current tuple (except where a tuple has been deleted, in which case it is moved to the history chain, and in its place a "delete indication" is kept). With this basic structure, all current and history data in a table can be accessed, although only sequentially. In order to allow random or direct access, a rather complicated indexing method is implemented using ordinary trees.

[Lum 84] also discusses the problem of the nature of time raised also in [Ariav 83a], and distinguishes between physical time, i.e., the time on the database interval clock, and the logical time, which is the time associated with the user's application perspective. This distinction is needed in situations like retroactive reporting, in which the use of only a single parameter for time will lead to the loss of information. the paper's conclusion is that the physical time will be used to record all database actions, and the user will decide whether to define explicitly one or more additional time parameters in the database. These "time" parameters are referred to as logical time, based on the real physical time as a reference.

The indexing methods in [Lum 84] are complicated, and cause the execution of many operations whenever a change is made in the database, in order to

maintain its consistency. The paper describes how to store and retrieve data, but does not cover the crucial aspect of operations on the data.

1.4.4. Implementations Without Theoretical Basis

There have been some attempts to develop data management systems that exhibit some of the properties addressed by the studies mentioned above. For instance, [Wiederhold 75] describes a medical database of temporal information, dubbed TOD (Time-Oriented Database).

The design of TOD distinguishes between the stable patient information and the volatile, chronologically ordered visit data, and organizes them in two separate files. This distinction is similar, in some ways, to the distinction adopted in our research, which differentiate between constant attributes (CAs), and time-varying attributes (VAs). In TOD, visit numbers are used to locate the records via the index structure of the historical file. The first and the last visit of every patient are specified in the patient's header record, and other visits in the parameter file are chained both forward and backwards, to facilitate efficient retrieval. The system is capable of retrieving information, both for tracing a specific patient's condition, and for statistical analysis across patients.

TOD is, in fact, a special purpose system, developed specifically for hospital environment. It handles temporal information, but the distinction between static data items and dynamic data items is not in the level of attributes. It is instead in a higher level of the general patients' information and their visits' information.

A more generalized version of a TDMS is the DATA system (for Dynamic Alerting Transaction Analysis system) [Ariav 81], a rudimentary, relational-like system that handles certain time aspects in an explicit manner. The data in the system is maintained as a cumulative, time ordered list of transactions (i.e., recorded events), and the status of an entity "as of time t" is derived from the collection of transactions relating to that entity which have been recorded prior to time t, and ignoring all transactions since t. The specialized commands for dealing with time are the following: rundown replays a sequence of recorded events between two points of time, settime allows the user to view the database from previous points in time, and rollback, backs up the database to permit operations on its contents at some previous point in time.

Transactions are always appended to the database, and are never modified or deleted. Nevertheless, they can be one of three basic types, namely addition, modification or deletion. Each transaction contains a time stamp, a pointer to the previous transactions related to the same entity, and eventually the new data value. This architecture actually extends the notion of "differential files" [Severance 76], to the point where the background database is simply an empty set.

The DATA system associates the time stamps with the whole tuples, and as such it suffers from the corresponding storage redundancy problem. In addition, it does not actually combine data of different objects to produce reports for the various potential users' views, but is only a tool to maintain the "complete history" in its transactional form, and retrieve it whenever required.

1.5. The Structure of this Research

This chapter has presented the background of this research, described the general nature and capabilities of a Temporally Oriented Database (TODB) and surveyed related literature. It then presented the major concept of the research - the temporal differentiation of attributes, and outlined the general research question, namely: how should we design and implement a TDMS based upon this concept.

Chapter 2 presents the general framework of this research, develops the specific research questions and describes the methodologies that were used to answer them.

Chapter 3 presents the basic data structures used in our model, using the concept of the temporal differentiation of attributes [Clifford 83b]. These data structures are stored in regular relations, allowing us to use an existing DBMS to implement and manipulate them. In so doing, we do not need to design the modules to carry out the I/O operations, and are able to concentrate on the conceptual components of the system. This chapter also describes the ways to create the TODB scheme, to load it with data and to update it, including the procedures to maintain the TODB's consistency. The simplicity of these tasks underscores one of the advantages of the proposed design.

Chapter 4 develops the general framework within which the temporal relational algebra operations are defined and implemented in this dissertation. Chapters 5, 6 and 7 cover the temporal relational algebra

operations, SELECT, PROJECT and JOIN correspondingly. The operations are explained through concrete TODB examples covering all their variations in the unique environment of TODBs. The detailed definitions of the temporal operations set this dissertation apart from previous attempts to study the implementation of TDMSs.

Chapter 8 contains a discussion of the results of this research, with an attempt to generalize the results and put them in a broader perspective. Chapter 9 concludes this dissertation, highlighting its contribution and significance and outlining the major research issues that emerge from our research.

Chapter 2

Research Goals and Methodologies

The focus of this research is the assessment of the temporal differentiation of attributes as an implementation strategy for relational TODBs.

2.1. The Research Framework

This section outlines the conceptual view of a TODB, and presents the overall framework of the research.

2.1.1. Basic Theoretical Components

The external data structure of TODBs is a cube, instead of a table in the traditional relational databases. The cube represents a Temporally Oriented Relation (TOR). An example of such a cube is presented in Figure 2-1.

All TORs have TIME as their third dimension, added to the two that already exist in regular relations. The time-unit is typically application dependent. Throughout this dissertation, the time-unit will be a day (denoted YYMMDD; e.g., 820517 means: May 17, 1982). In other applications it may be an hour, a month, or any time-unit that fits the nature of the database. However, only one time-unit is allowed in one TODB. The TDMS, in fact, does not "know" the meaning of the numbers representing the time stamps for the

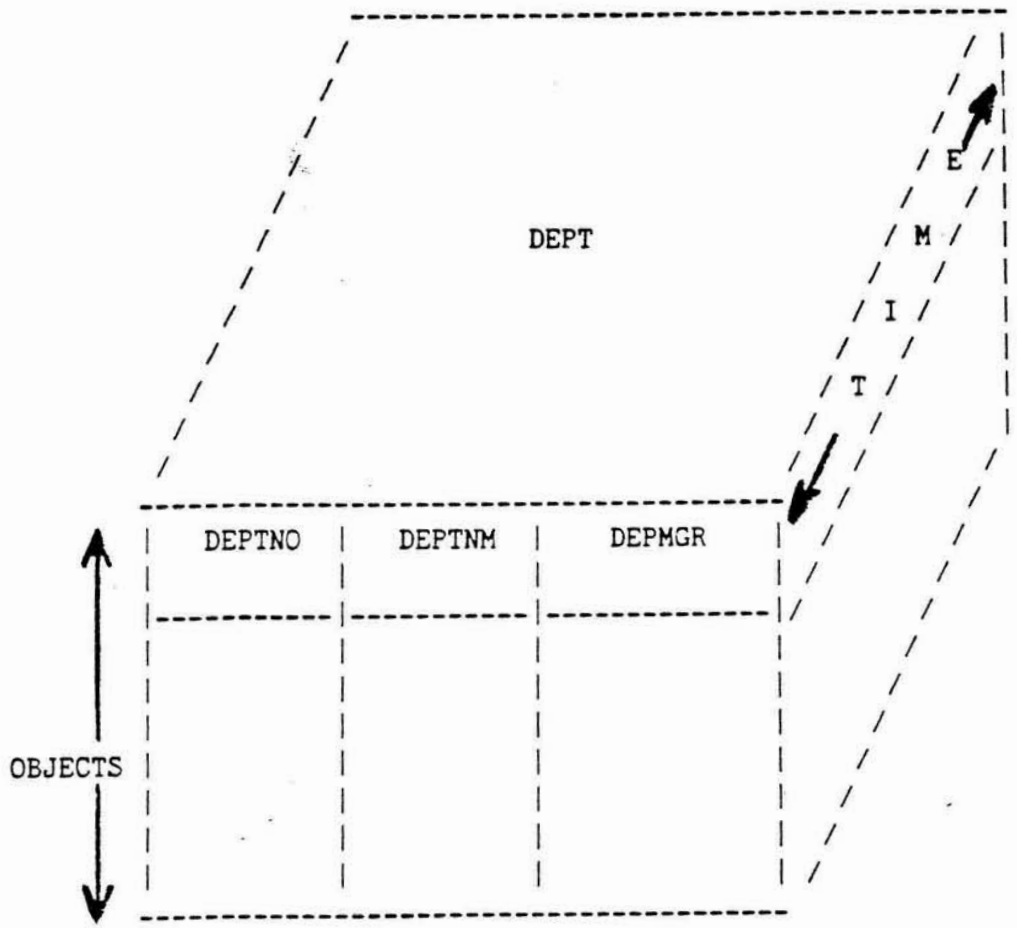

Figure 2-1: The Basic Data Structure of TODBs (the Cube)

attributes in the various TORs. Consequently, it cannot handle the existence of different time-units, and execute operations requiring an interpretation of their relationship. This self-imposed limitation enables us to concentrate on other issues in a TDMS. Further research should explore the structure of the time domain, operations on that structure, relationships between time domains, etc.

The cube representing the TOR actually contains one time slice for each

day, containing the data for this particular day. Let us assume the existence of the TOR DEPT, with values for all attributes at each day, starting, for instance, at 800101. The data for two arbitrary days in this cube is shown in Table 2-1 (the source data for these time slices is described later).

The data for 801010:

DEPTNO	DEPTNM	DEPMGR
1	SALES	10050
2	PRODUCTION	10030
3	ACCOUNTING	10010
4	MANAGEMENT	10025

The data for 831230:

DEPTNO	DEPTNM	DEPMGR
1	SALES	10050
2	PRODUCTION	10005
3	ACCOUNTING	10080
4	MANAGEMENT	10025

Table 2-1: Two Time Slices from the Cube Describing the TOR DEPT

In general, we will use the convention that two TORs are considered to be equivalent during a time interval, if all their time slices for the same time points are equal.

In a regular relation, the information about an object is included in a

tuple of this relation. In a TOR, however, every object has a tuple for each time-point, creating together a horizontal slice of the cube. "Tuple" in a TOR has the same meaning as "tuple" in a regular relation, but while an ordinary tuple in the regular case describes the current status of the object, a tuple in a TOR contains the information of the object at a specific time point. This time-point, however, is not included in the tuple, since time is an implicit entity of the TODB, and does not appear in its TOR's schemes.

Conceptually, there is potentially a different time slice (table) in each TOR for each day, starting at 800101. All the time slices of one TOR conceptually constitute its cubic view. Needless to say, it is extremely wasteful to actually store the data this way. Therefore, another view of the data, an internal view, is defined as the underlying functional view of the data, that contains only the minimal amount of data needed to create the external user's view.

The internal view [Clifford 83b] takes advantage of the differentiation between constant attributes (CAs) which are time invariant, and time-varying attributes (VAs) that do change along time. In this internal view, each time-varying attribute (VA), such as SALARY, has as its domain not simple values (such as 25,000) but rather functions from time points to values. The representation of a VA by the minimal data needed to determine its value for each time-point is called a partial specification of the VA, since by itself it provides only partial information about the values of this VA in this TOR. This information is indeed sufficient for determining the values of this VA for any object at any point of time, since there is a known interpolation

mechanism that completes the description of the data. The internal view of a TOR, therefore, contains the data for all its CAs and the partial specifications of its VAs.

The operations on the above data constructs could be viewed as expanded definitions of the traditional relational algebra operations, forming a temporal relational algebra. These operations create new TORs, thus maintaining the "closure" property of the relational model. In general, we strive to define the temporal operations as a consistent extension of the regular relational algebra operations, and thereby maintain the meaning of a regular operation when applied to a TOR that contains equivalent of a single time slice.

2.1.2. The General Framework of the Research

The framework of the design of our TDMS is depicted as in Figure 2-2. The external view in this framework is the cube which is the basic structure as viewed by the user for representing temporal data. The internal view contains the minimal data needed to simulate the cube. The values of the CAs in this view are atomic values, while the VAs contain functions from time-points to values, for those time-points in which values have changed. The internal view is not convenient for storage in a traditional relational DBMS. Therefore, we rearrange the information included in this internal view, and store it in regular relations (The exact structure is presented in Chapter 3). One should note that the information recorded in these representing relations is identical to the information included in the internal view; the same is true for the mechanisms to construct the whole cube out of either the internal

view or the representing relations. Another important point is that storing the information in these representing relations allows us to manipulate them by an existing relational DBMS. The interpretation of these relations as representing the TORs in the TODB is carried out by the TDMS. Both the internal view and the representing relations can be mapped to the physical storage which is the lowest level in the research framework, that concerns the actual storage of data. In general, we will deal only with the higher levels of the framework; specifically, the TOR and its underlying representing relations.

In TDMS, the user interfaces with the database by using the terms of the external view, and can generally be ignorant of the distinction between CAs and VAs. However, these two different kinds of attributes are stored and handled differently in our model. Therefore, each attribute must be classified to the TDMS upon the creation of a new TOR. Thus, the database administrator (DBA) should be aware of this issue, and whenever he wishes to create a new TOR, he has to specify its key, its CAs and its VAs. From this point on, this knowledge exists in the TODB, and the user never has to repeat it. The levels below the internal view are completely hidden from the user, and he never deals with them or has to provide information for them.

The major effort in developing TDMS is the formulation of temporal operations, conceptually defined on the external cubic view(s), but actually executed on the representing relations. Therefore, in developing these operations we are concerned with consistent mapping from the cubic view(s) to the representing relations. The internal view is mainly used to present the

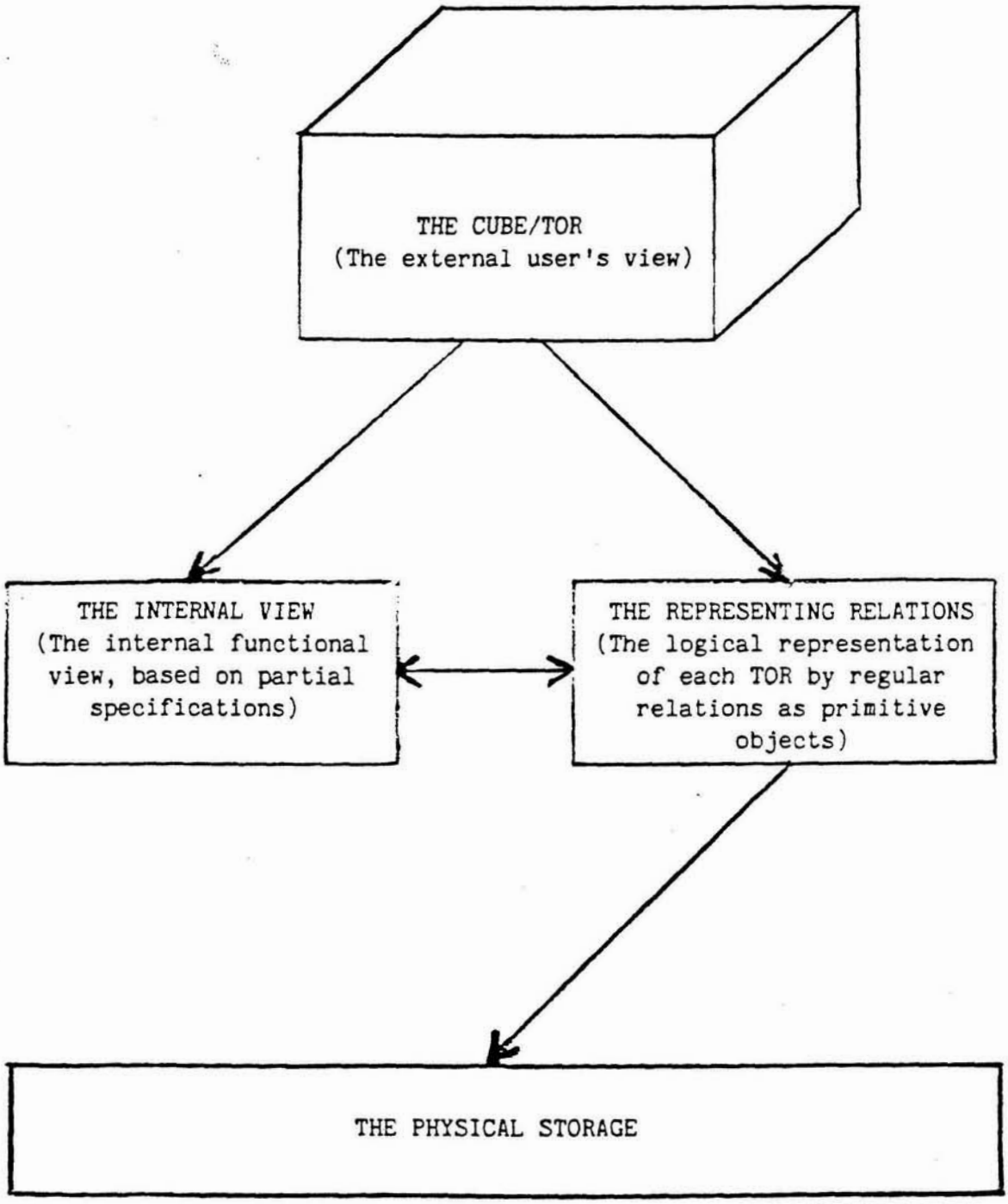

Figure 2-2: The Framework of the Research

results of the various examples, raised during the discussions about these operations in Chapter 5 through 7 below. The internal view seems preferable for presenting The results to the user, while the representing relations are essential to the execution of the operations in our implementation approach. As the two are equivalent, we use the representing relations in the detailed design of the temporal operations, while using the internal view for presentation purposes. It should be noted that a different implementation strategy could have used the internal view as the underlying data structure.

2.2. The Issues Involved in Implementing TODBs

As the literature survey indicates, implementing TODBs involves a whole set of design issues. A critical survey of the previous research on these subjects suggests the major topics highlighted in the following paragraphs. Later in this chapter, we identify those implementation issues on which the dissertation centers.

* The types of time

  When dealing with time, one may consider different types of time stamps that could be associated with data items, e.g., the time in which some event happened, the recording time, etc. This issue has been dealt with in several papers, e.g., [Ariav 83a], [Lum 84], [Snodgrass 85], and introduces a higher level of complexity into temporal data models.

* Data structures

  A number of researchers (e.g., [Wiederhold 75], [Clifford 82a], [Ariav 83a], [Lum 84] among others) view temporally oriented data conceptually as a three-dimensional cube. It is clear that the cube contains extremely redundant information, and the data cannot possibly be stored as such. Therefore, a primary design decision in implementing a TODB concerns its data structures and storage methods. They should provide a method to store the minimal amount

of data needed to construct the whole cube, in a way that is efficient both for storage and for retrieval.

* Temporal operations

The capability to retrieve any desired information from an existing TOR is just the first step in developing TDMSs. Experience with regular databases has already taught us that the relation schemes should be designed to conform to the highest normal forms, thus preventing unnecessary redundancy in data, and various anomalies in updating operations. The price for this design is that many common queries cannot be answered directly from single relations, but require the creation of new (possibly temporary) relations, using relational algebra operations. In TODBs, we face the same problem, and we cannot expect to have all desired queries answered by using only single TORs. We therefore have to add more functionality to the system, and the next step in building a TDMS is the definition and implementation of operations on the TORs, in order to accomodate more advanced queries that people are likely to ask.

* Integrity constraints

Any DBMS should ideally provide tools to maintain consistency in the database. This issue is probably more important in TDMS which handles objects with many different tuples for the various time points associated with their attributes. Another factor affecting the importance of this point in the TODB is that data is never modified or deleted, a fact that should dictate the execution of a comprehensive validation procedure before a data item is allowed to enter the TODB.

* Implementation approaches

In the design and implementation of a TDMS that extends the relational model, there are two basic approaches, as suggested by [Ben-Zvi 82]:

1. Starting from scratch, and devising all the needed modules, including the I/O module, internal manipulations, user interface, etc.

2. Using an existing DBMS as a tool to manipulate the TORs as needed. On top of this DBMS, an interpretive mechanism could be built to mediate between the user and the DBMS. The user then deals with external views, the cubes, while the DBMS contains regular relations. The modules added to the DBMS fill this gap between the user's view and the DBMS's content.

A product obtained with the first approach is likely to be superior in performance, but would probably take much longer to develop, especially if it contains all the desired properties that already exist in commercial DBMSs.

In the second approach, one is relieved from dealing with I/O operations and file management, since a reference to the DBMS will be made whenever needed, handling a tuple at a time from regular relations to manipulate them as desired. Therefore, the completion of a TDMS prototype would be much easier, even though its performance may be inferior.

* Query languages and user interfaces

A TDMS requires a special kind of query language to support questions referring to the temporal dimension of the data (e.g., the language suggested by [Ariav 85] or [Snodgrass 84]), containing the necessary extensions for the specifications of the query's time aspects.

The issue of query languages raises the aspect of interfaces. The issue of friendly user interfaces becomes more complicated in TDMSs, because of the added temporal dimension.

* Query optimization

A query optimization procedure is a very important component of a DBMS such as INGRES, that uses a non-procedural query language, and may be even more important in TDMSs, since each single temporal operation requires the execution of many computations, and the combination of several operations needed to answer more advanced queries may be very expensive.

* Temporally oriented (evolving) schemes

In the course of the life of a database, it may happen that modifications to some relation scheme are needed, e.g., a new attribute is added or dropped ([Navathe 80], [Ariav 83a]). So, for example, even though there are no past values for the new attribute, the user may wish to start managing it, together with the other attributes, from now on. This implies the conversion of the concept of a scheme from a constant entity to a time-varying one. Thus, a TOR can have different schemes for different periods of time, and whenever a modification to the scheme is needed, another version of it is created. The new scheme should not require any duplication of data, but should refer differently to the already existing data, and

possibly combine it with new attribute(s) added to the scheme in this version. A new version of the scheme does not cancel the old ones; the system should determine, based on the user's query, which version(s) of the scheme to apply.

The capability to handle evolving schemes is a natural outgrowth of research on historical databases. Evolving schema allow the same TOR to include different combinations of attributes in different periods of time, thus introducing an additional and very powerful flexibility in the ways that data can be viewed in different periods of time. Such an evolving schemes capability is, of course, a very desirable property of a TDMS. This issue has been initially discussed in [Ariav 83a], but because of its complexity it is not yet clear if and how it can be incorporated into an implementation of a TDMS.

2.3. The Research Questions

The basic principle of the temporal differentiation of attributes has implications for every issue involved in implementing TODBs. A relatively closed subset of issues has been designated as the set of issues to be addressed in this research. This dissertation research examines therefore the impact of temporal differentiation of attributes on the design and implementation of TODBs. This section outlines the specific research questions addressed in this dissertation. There are other issues involved in implementing TODBs with which we do not deal in this research. We comment on some of them in the conclusion of this research, pointing out the potential benefit of our basic concept as applied to them.

Before presenting the research questions, a comment on the role of time in TODBs is due. The precise meanining of time in temporally oriented databases has not been well defined in many of the research efforts proposing schemes to satisfy the need for supporting time varying information in

database systems. Furthermore, there has been confusion concerning the terminology and definition of the time attributes in many of these research efforts [Snodgrass 85].

In our data model the time values associated with the various data items in the TODB specify the starting validity time of these data items, and have nothing to do with other possible time perspectives, such as recording time. Dealing with more than one time dimension requires multiple time-stamps associated with each data item (see [Ariav 83a]), and is not covered by this research.

We believe that one cannot resolve the problem of databases with multiple-time dimensions, without having first a single-time TDMS such as the one studied in this research. Therefore, solving the problem of handling historical databases arranged along a single dimension of time is an essential step towards the development of more complicated databases. The proper treatment of more than one temporal dimension is clearly a subject of future research.

All the research questions deal with the temporal operations. Extending the existing relational algebra operations into temporal relational algebra operations is a complicated effort. It immediately raises many questions, among which the major are:

* Research question 1

  Is there a "natural" extension of the regular relational algebra operations into temporal relational algebra operations? Could such an extension maintain the closure property?

* Research question 2

  How are the algorithmic definitions of these operations affected by the types of attributes being manipulated, namely: key attributes, CAs (constant attributes) or VAs (time-varying attribute)? How are the results affected by the types of the involved attributes?

* Research question 3

  How is the time dimension of the result inherited from the time dimension(s) of the operand(s)?

These questions about the temporal relational algebra operations may not exhaust all the problems concerning them, but seem to be central ones and reflecting the complexity of the issues on hand.

2.4. Research Methodologies

All the research questions stated above refer to the impact of the temporal differentiation of attributes on the operations in TDMSs. These questions can conveniently be addressed through the development of a detailed design of a TDMS, and the application of this TDMS in the management of a benchmark database.

2.4.1. Detailed Design of the TDMS

A major part of the research has been a complete design of a TDMS. This entailed the definition of a mapping between the conceptual view of data (the cube) and the primitive objects (the representing relations) that are used to implement it. This mapping exploits the temporal differentiation of attributes, and constitutes an implementation-level data model. The detailed design contains the full set of algorithms in a TDMS in sufficient detail to program them. The actual programming of these algorithms and application in

various examples serves as a quality control, and provides the means to identify implementation problems and a vehicle for the demonstration of the various features of such a system. The design provides a basic means for assessing the usefulness of this concept as an implementation strategy for TDMSs.

The design details how to store the minimal amount of information needed to create the whole cube. It demonstrates that efficient storage can be accomplished by using regular relations as primitive objects, without any additional types of data structures. The full design shows exactly what the data structures are, how they are maintained, and what mapping techniques are used to retrieve data from them, in order to simulate the user's cubic view.

Research question 1 deals with the definition of operations needed to answer user queries. The design defines such operations as natural extensions of the regular relational algebra operations. These definitions maintain the closure property, so that each operation creates a new valid TOR. The definitions of these temporal relational algebra operations refer conceptually to the cubes, but the algorithmic definitions deal with our data structures. They manipulate a sequence of regular relations representing the operand(s), perform operations on these relations, and create a new sequence of regular relations representing the resulting cube. A full mapping among the external and the implementation data model demonstrates the correctness of the algorithmic definitions.

While developing the algorithmic definitions of the temporal operations,

we systematically analyze the effect of the types of participating attributes. Special care is given to the temporal JOIN operation, which is the most complicated operation in TDMSs. Our analysis of the temporal operations includes many examples, demonstrating the impact of the various attributes types involved in these operations on their results. This responds to research question 2.

The result of any temporal operation is a new TOR. Consequently, it contains time stamps associated with its attributes. The design of the operation must specify how these time stamps are inherited from the time stamps associated with the attributes of the operand(s). This analysis is part of the design of the temporal relational algebra operations, and therefore answers research question 3.

2.4.2. Use of Actual Database

A benchmark database was used to investigate the nature of a TODB. Discussing detailed concrete examples is one way to demonstrate the specifics of the design and the implementation, to present the internal representations of the TORs, and to show how the representing relations are manipulated to create new TORs as results of temporal operations.

We have chosen a personnel database containing nine TORs (see Figure 2-3), each of which has at least one time-varying attribute. This database is much richer than the examples used in previous works like [Ariav 83c], [Clifford 82a], and [Snodgrass 84], and the variety of its TORs and their attributes allowed us to study the intricate details to be resolved in TDMS

design. It also enables us to assess many logical and practical queries whose answers demand the execution of temporal relational algebra operations. These queries are used to illustrate the problems in our design, and help outline their solutions.

---

* EMP(EMPNO,NAME,SEX,DEPTNO,JOBCLS)
* DEPT(DEPTNO,DEPTNM,DEPMGR)
* SAL(EMPNO,SALARY)
* COURSE(CRSNO,CNAME,PRICE,DURATN)
* TRNHST(EMPNO,CRSNO,GRADE)
* DRESS(SEX,ROOM)
* UNIONS(UNION,SEX,OFFICE)
* PHONES(PHONE,DEPTNO,LINES)
* PROJECTS(PROJNO,PROJNM,COST,DEPTNO)

(key attributes are underlined)

Figure 2-3: The TORs in the Benchmark Database

The benchmark database is used throughout the dissertation. In the following paragraphs we briefly introduce its structure and content.

The relation EMP contains the basic information about each employee: his/her identification number (EMPNO), NAME, SEX (M or F), the department number (DEPTNO) to which he/she is assigned, and his/her JOBCLS (a number indicating class of occupation).

The relation DEPT describes the departments in this firm. DEPTNO is the department number, DEPTNM is its name, and DEPMGR is the identification number (EMPNO) of its manager.

The relation SAL holds the salary, SAL, of each employee. In fact, it can be merged with the relation EMP, but it has been defined separately, to enable us to illustrate later the simplest possible JOIN operation in our TDMS. It contains the employee identification number (EMPNO) and his SALARY.

The relation COURSE describes the courses given within the training system of the firm. CRSNO is the course-number, CNAME is its name, PRICE is the price charged for it, and DURATN is its duration in days.

The relation TRNHST describes the instruction-history of each employee, and contains his/her EMPNO, the number of the course taken (CRSNO), and the GRADE achieved. Note that in this relation the key consists of two attributes, EMPNO and CRSNO.

The relation DRESS stores the dressing room allocated to the male employees and to the female employees in the organization. It contains the key SEX, and the allocated dressing ROOM. The relation UNIONS describes the various unions present in the company. There are several unions, each of which is open either only to men, or only to women. The relation contains the union name (UNION), the SEX to which it is open, and its OFFICE.

The relation PHONES describes the phone lines allocated to each department in the organization. The key of this relation, PHONE, is a

telephone number, DEPTNO is the department number to which the telephone is assigned, and LINES is the number of lines allocated to this phone number.

Finally, the relation PROJECTS describes the various projects handled by the company. Its key is the project number PROJNO, PROJNM is the project name, COST is the estimated cost of the project, and DEPTNO is the identifier of the department responsible for this project.

An examination of the attributes included in this database reveals that some of them, like EMPNO, NAME and SEX in the relation EMP, and CNAME in the relation COURSE, are not subject to changes along time, while others, like JOBCLS in the relation EMP, and SALARY in the relation SAL, do change at various points of time.

2.5. Summary

Adopting the concept of the temporal differentiation of attributes raises several questions concerning the impact of this concept on the design of TDMSs. This chapter presented the specific research questions and the methodologies used to answer them. These methodologies center around a complete design of a TDMS, based on the data model developed in this dissertation. Like every other data model, ours contains the three major components: data structures, constraints and operations. They are discussed in full detail within the next chapters of the dissertation.

Chapter 3

Data Structures in TODBs

This research develops a data model to implement temporally oriented relational databases, and as such, it contains all three major components of every data model: data structures, operations and constraints. This chapter covers two of our model's components, the data structures and the integrity constraints. Their design is the background for the discussions in chapters 4 through 7 that deal with the research questions presented in Chapter 2.

3.1. The External Cubic View

The conceptual objects handled by a TDMS are cubes of data rather than flat tables, and in our terminology we refer to them as Temporally-Oriented Relations (TORs). A typical cube is illustrated in figure 3-1, representing the TOR EMP as viewed by the user. The cube is conceptually constructed by the collection of all time slices corresponding to all time points (days in our example) included in it. Table 3-1 presents the flat tables containing the data of two arbitrary days in the TOR EMP. The source data for these time slices is included in Appendix A.

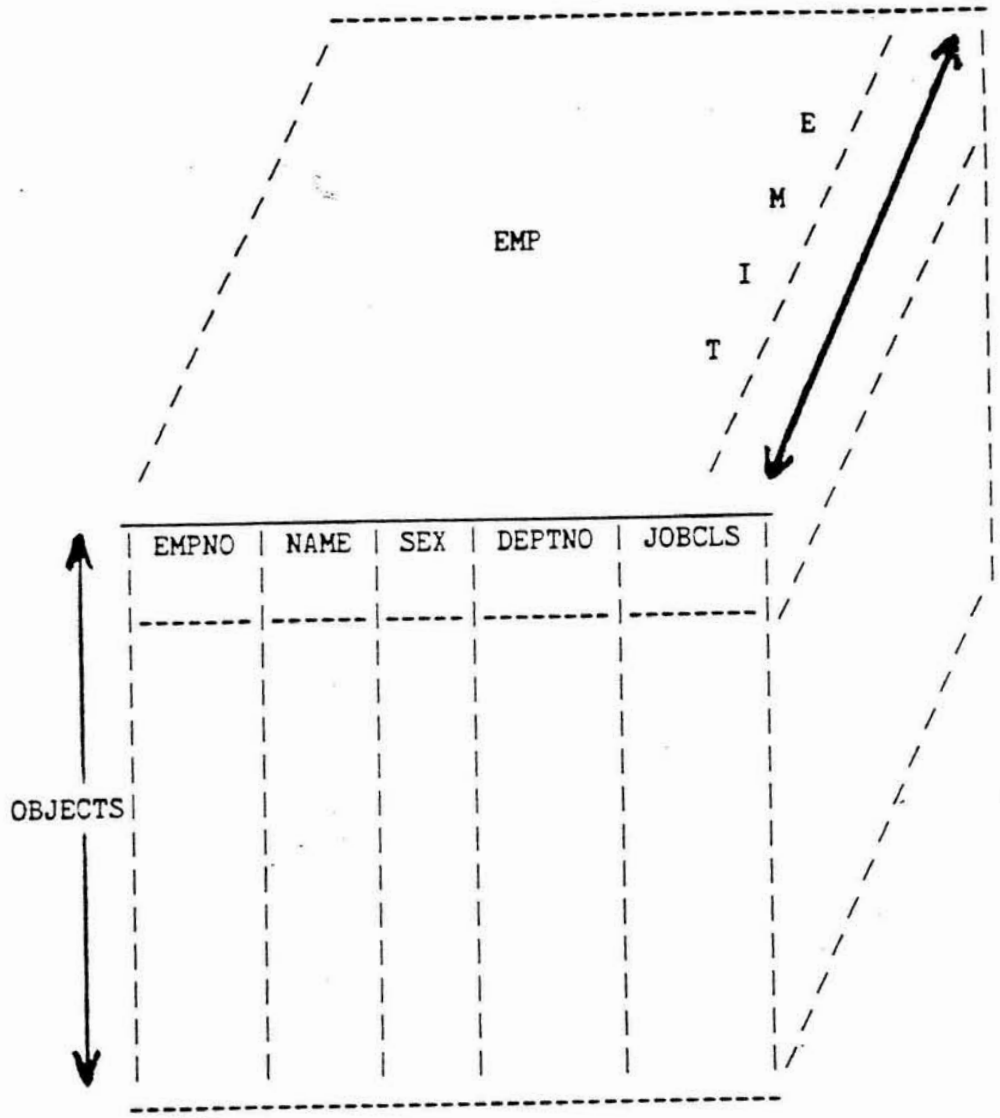

Figure 3-1: The User's External Cubic View of a TOR

For 800101:

EMPNO	NAME	SEX	DEPTNO	JOBCLS
10010	MIKE	M	3	4
10090	SUSAN	F	4	4
10030	HENRY	M	2	3
10025	OSCAR	M	4	1

For 811020:

EMPNO	NAME	SEX	DEPTNO	JOBCLS
10010	MIKE	M	2	3
10090	SUSAN	F	4	3
10030	HENRY	M	2	3
10025	OSCAR	M	4	1
10005	MARY	F	2	3
10050	DAVID	M	1	3
10080	ALICE	F	3	2

Table 3-1: Examples of Time Slices from the TOR EMP

Each of the time slices constructing the cube corresponds to one time point, and is a valid regular relation. In order to understand the conceptual "construction" of the cube, one can imagine that for some relational scheme, data is recorded at every time point (in our example, every day), creating a separate relation containing this day's data. Then, all these relations are combined together sequentially. Each of these time slices, being a regular relation, consists of tuples representing its various objects. "Tuple" in the cube is the same as "tuple" in a regular relation. It contains values for all

the attributes included in the cube's scheme for a specific object at a specific time point. In the regular relational model, a tuple contains the current values of the particular object. Whenever one of these values changes, the new value replaces the old one, causing the loss of the information just replaced. In our external cubic view, a new relation is created for each time point, containing a tuple for each individual object. This relation "exists" conceptually even if it is identical to the one that immediately preceeded it. An object, therefore, is conceptually represented in the cube by a horizontal slice (horizontal layer), containing a separate tuple for this object's values at each time point.

The three dimensions of the cube are the following: the objects dimension, the attributes dimension, and the time dimension. The first two play in our model the same role as in the regular relational model. The third dimension, TIME, starts at some time point in the past, at which the data collection started, and contains all time points from then until now.

As the cube is conceptually constructed of the time slices corresponding to the different time points on the time axis, it is useful to take a closer look at these time slices. There are objects (e.g., employee 10090 in Table 3-1) whose data are identical in the two time-slices, and possibly identical in all time slices corresponding to dates between 800101 and 811020. There are attributes that by their nature do not change at all over time (e.g., EMPNO, NAME and SEX in EMP; DEPTNM in DEPT), while others (e.g., DEPTNO and JOBCLS in EMP; GRADE in TRNHST) do. Moreover, the number of objects does change over time, i.e., employees are hired or may quit. Quiting means that

their records do not "exists" in the database after their quitting dates, even though their data is never deleted.

All these observations raise several problems concerning this external cubic view:

1. There are attributes that do not change at all over time, and repeating them may create an enormous amount of redundant stored data in the cube.

2. Even some time varying attributes (such as DEPTNO in EMP and GRADE in TRNHST) do not change very frequently, and therefore introduce further redundancy into the database.

3. There is a problem with representing the fact that some objects do not exist in various time-points of the database, while they do in others. This problem is relevant, for example, to employee 10050 in the TOR EMP.

3.2. The Internal View

As a solution to the problems raised in the previous section, we propose the definition of an internal view of the TOR, which contains only the minimal amount of information needed to construct the whole cube, by associating time with each attribute in the TOR [Clifford 83b] and [Clifford 85a], rather than with a whole tuple, which has been the common practice so far. This approach provides more flexibility in designing the internal view, allowing each attribute to change along time independently of other attributes in the same TOR.

The components of the internal view are:

1. The internal view distinguishes between changing and non-changing attributes. Attributes that do not change along time (like NAME, SEX, SOCIAL-SECURITY-NUMBER) are called constant-attributes (CAs).

Attributes that change along time are called time-varying attributes (VAs).

2. The domain of a CA contains atomic values (names, numbers, etc.) just like attributes in regular relations.

3. The domain of a VA contains functions instead of values. Each object has its own function from time-points to the actual values [Klopprogge 81]. There are, of course, many time-points in each of these functions, with a lot of redundancy in recording all of them. Therefore, following [Clifford 83b] and [Clifford 85a], we define a partial specification for every VA. It contains only those time-points in its function, at which the value has changed. The partial specification of a VA contains the minimal amount of data needed to determine its values at all time-points. As an example, see Table 3-2, presenting the partial specification of the VA DEPTNO for object 10030 in the TOR EMP.

TIME	DEPTNO
800101	2
820701	3
830508	2

Table 3-2: A Typical Partial Specification Describing a VA

4. A NULL value is used in the partial specification of a VA of a specific object in some TOR at some time point, whenever no other value, taken from this VA's domain, is known to prevail for this time point. The use of a NULL value may be caused by many reasons, and the detailed discussion about them is beyond the scope of this research [Clifford 85a]. Their most common use, however, is to indicate the absence of a value for a VA in an object of some TOR, at a specific time point. Inexistence of an object during a given period of time is indicated in our model by assigning NULL values to all its VAs during this period. This is the main use of NULL values in our model. These NULL values are automatically in effect until other assignments are made to the object's VAs. Therefore, whenever a query, refering to time points in this time interval, is executed, this object shows up in the answer with NULL values assigned to all its VAs, indicating its non-existence. As one of the basic

> principles underlying the TODB is that data is never deleted or modified, no previous data of this object is affected by the fact that, starting at some time point, this object no longer exists in the TOR (e.g., an employee who quits his job). Included in this previous data are all its CAs, including its key. They are left unchanged even during the period in which the object does not exist, since by their nature they never change.

The use of NULL values assigned to all VAs of an object at a specific time point as indicator of non-existence affects the definitions of operations later on. Every temporal operation creates a new TOR, that may have objects that do not exist at some periods of time, as inherited from the operand(s). The procedures to execute those operations should contain the steps to maintain consistency in representing such situations in the resulting TORs. Such steps would have been needed in any other method used to reflect non-existence of objects at some time points, since this situation is an integral part of the TODB's nature. As we found out, our method is sufficient for maintaining consistency in all the operations executed with TORs. However, other methods can be suggested, and future research should deal with the use of NULL values in TODBs.

The partial specification of a VA of some object in a TOR contains only values for time points at which these values have changed. Therefore, an interpolation function is also needed, to give the full function definition [Clifford 83b] and [Clifford 85a]. It is clear that by itself, without an interpolation function, the partial specification indeed gives only partial information about the values of the VA. This interpolation function is associated with each VA in each TOR, and enables us to determine the value of

this VA at time-points in three distinct periods of time, i.e., (1) before the first explicitly specified time-point, (2) after the last time-point specified, and (3) between any two time-points within the partial specification of the VA. For instance, for the VA DEPTNO in the TOR EMP, the following step function interpolation could be used:

* For a time-point before the first one, the value of the function is NULL.

* For a time-point after the first one the value is equal to the value associated with the latest specified entry before that time-point.

Note that the only property of time that is relied upon to provide this definition is that it is totally ordered (i.e., for any two points t_1 and t_2 either $t_1 < t_2$, $t_2 < t_1$, or $t_1 = t_2$).

The definition of this function reflects the typical interpolation for this attribute. However, there may be other interpolation functions according to the nature of the VA, e.g., for the VA "temperature of a patient" in a hospital, the value at a time-point between two existing values can be their average, or another linear combination of them.

The partial specification, defined above, is the collection of all explicitly recorded values in a VA. It can easily be seen that a partial specification of a VA, together with its interpolation function, gives complete information about it, and enables the determination of its values at all time-points.

The internal view of a TOR is the representation of this TOR by means of

its CAs and the partial specifications of its VAs, together with their interpolation functions. Note that the constant attributes in this internal view contain (like attributes in regular relations) atomic values rather than functions. The internal views of the TORs EMP and DEPT are presented in Tables 3-3 and 3-4 respectively.

EMP(EMPNO,NAME,SEX,DEPTNO,JOBCLS)

			CAs	VAs
EMPNO	NAME	SEX	DEPTNO	JOBCLS
10010	MIKE	M	800101 3 810215 2	800101 4 810201 3 821015 2
10005	MARY	F	810210 2	810210 3
10050	DAVID	M	800601 1 820508 NULL 830415 1	800601 3 820508 NULL 830415 2
10030	HENRY	M	800101 2 820701 3 830508 2	800101 3 820101 2 830304 1
10080	ALICE	F	810101 3	810101 2
10025	OSCAR	M	800101 4	800101 1
10090	SUSAN	F	800101 4	800101 4 811015 3

Table 3-3: The Internal View of the TOR EMP

Tables 3-3 and 3-4 demonstrate the use of NULL values. Interpreting Table 3-3, using a step interpolation function for both DEPTNO and JOBCLS, leads to

DEPT(DEPTNO,DEPTNM,DEPMGR)

DEPTNO	DEPTNM	DEPMGR
1	SALES	800601 10050
		820508 NULL
		830415 10050
2	PRODUCTION	800101 10030
		820701 10005
3	ACCOUNTING	800101 10010
		810215 10080
4	MANAGEMENT	800101 10025

Table 3-4: The Internal View of the TOR DEPT

the observation that all the VAs of object (employee) 10050 during the period 820508 - 830414 are NULL, indicating that this employee actually did not exist in the organization during that period. The "story" behind this information is that this employee had actually quit his job in the organization at 820508, and then returned back at 830415.

The rather intricate internal view is hidden from the user whose view is the whole cube, containing a full table of values for every point in time. However, in the first step of creating a TOR, the DBA has to classify each attribute in it as one of the following categories: a key attribute, a non-key constant attribute, or a VA, and specify the interpolation function for each of its VAs. This information is needed in building the internal view which is the basis for creating the physically implemented TODB, represented by regular relations.

The property of being a CA or a VA is associated with an attribute within a specific TOR. For example, the attribute DEPTNO in the TOR EMP is a VA, while it is the key of the TOR DEPT.

Assuming that a cube summarizes the information contained in its entire set of time slices, we can define the relationships between the two through the steps necessary to build the cube using this sequence of time slices, namely:

1. Scan all the relations, and prepare a list of all different objects appearing in at least one of them.

2. Record the CAs of these objects in the cube at all time points. These values for a specific object can be copied from any relation containing this object, since they never change, and therefore are identical in all the relations containing it.

3. Scan the relations again, and copy the VAs of each object in each relation to the appropriate tuple in the cube (namely, to the tuple describing this object at the time point corresponding to this relation).

4. The previous step leaves possible "holes" for the VAs of the various objects at those time points in which these objects do not exist, and therefore these VAs could not possibly be copied from the relations corresponding to those time points. All these "holes" should be filled with NULL values that indicate the non-existence of an object at a specific time point.

This procedure clarifies the relationship between the cube and the various relations corresponding to all its time points.

After the conceptual creation of the cube, one can take a time slice from it at any time point, and compare it to the relation originally containing the data at this time point. In so doing, one will probably discover that this

time slice may not be absolutely identical to this relation. The relation is a subset of the time slice. The time slice may also contain tuples in which all the VAs are NULL. These tuples represent objects that do not exist at this time point. Therefore, they are not included at all in the original relation, but do belong to the cube. Nevertheless, the time slice is informationally equivalent to the original relation. This discussion suggests the following definition for the equivalence of two time slices:

Two time slices are equivalent if they contain the same tuples, except some possible tuples in either of them, containing NULL values for all their VAs. These tuples are practically ignorable, and therefore the two time slices are, in fact, equivalent. It should be noted, however, that these tuples add some information about the cube, namely, they indicate what objects exist in the cube at other time points.

Following the definition of equivalence between two time slices, this is the definition of equivalent cubes:

Two cubes, representing two TORs, are equivalent if all their time slices are equivalent.

These definitions are later used in this dissertation to demonstrate the correctness of TDMS's operations. They are based on the conceptual construction of the cube. However, one should not confuse between this conceptual construction of the cube with its actual representation in memory.

3.3. The TODB for the Benchmark Database

Recognizing the distinction between constant attributes (CAs), and time-varying attributes (VAs), we can now present our benchmark TODB. it is a temporal extension to the regular relational database presented in Chapter 2. This TODB is included in Figure 3-2.

---

* EMP (CAs: EMPNO,NAME,SEX ; VAs: DEPTNO,JOBCLS)
* DEPT (CAs: DEPTNO,DEPTNM ; VA: DEPMGR)
* SAL (CA: EMPNO ; VA: SALARY)
* COURSE (CAs: CRSNO,CNAME ; VAs: PRICE,DURATN)
* TRNHST (CAs: EMPNO,CRSNO ; VA: GRADE)
* DRESS (CA: SEX ; VA: ROOM)
* UNIONS (CAs: UNION,SEX ; VA: OFFICE)
* PHONES (CAs: PHONE,DEPTNO ; VA: LINES)
* PROJECTS (CAs: PROJNO,PROJNM ; VAs: COST,DEPTNO)

For simplicity, we assume that the interpolation functions of each VA in each TOR is a step-function.

Figure 3-2: The Benchmark TODB

3.4. The Underlying Storage Structures

The internal views do not address directly the data structures necessary to actually store the information. The actual representation of a TOR by a sequence of regular relations is explained in this section. Using regular relations as primitive objects allow us to bridge between TDMS and existing relational DBMSs that could be used to manipulate these regular relations as needed. As stated earlier, such a DBMS, INGRES, is used to implement the TDMS prototype as a part of this dissertation, saving, thereby, the effort of building the models to maintain and manipulate these regular relations.

The following paragraphs describe the relations that represent a TOR. One relation in this sequence contains all CAs in the TOR. The key of this relation is the key of the original TOR. This relation for the TOR EMP is included in Table 3-5.

EMP1(EMPNO,NAME,SEX)

EMPNO	NAME	SEX
10010	MIKE	M
10005	MARY	F
10050	DAVID	M
10030	HENRY	M
10080	ALICE	F
10025	OSCAR	M
10090	SUSAN	F

Table 3-5: The Relation Representing the CAs in EMP

One additional relation is created for each VA in the TOR. The

attributes in this relation consist of all the attributes in the key of the TOR, the attribute TIME, and finally the VA itself. The values stored for the VA are precisely its partial specification. The key of this relation contains all the attributes in the key of the TOR, together with the attribute TIME. The regular relations representing the VAs in the TOR EMP in our TODB are included in Table 3-6.

In addition to these relations that represent the actual data, we need another relation to hold certain meta-data, namely information needed to associate a cube (the external view) with the entire set of relations that store its content. This relation of meta-data describes the scheme of the TOR, as well as the types of its attributes. Specifically, this relation has the name of the TOR, and contains the attributes: ATTRIBUTE, PTYPE and LTYPE. All attributes belonging to the TOR are the objects of this relation. PTYPE contains their physical types (INTEGER, REAL, or CHARACTER), and LTYPE contains their logical types as follows: 1 for key attribute, 2 for constant non-key attribute, and 3 for time-varying attribute. Note that the system could be extended later to handle additional types of attributes, to indicate non-interpolatable time-varying attributes, to specify the interpolation function to be used, etc. The relation describing the scheme of the TOR EMP is included in Table 3-7.

In general, the meta-data relation of the TOR REL is REL, the relation describing the CAs of this TOR is REL_1, and, assuming that this TOR has n VAs, their corresponding relations are: REL_2, REL_3,...,REl_{n+1}. The names of the relations, associated with the various VAs, are determined by the order in

EMP2(EMPNO,TIME,DEPTNO)

EMPNO	TIME	DEPTNO
10010	800101	3
10010	810215	2
10005	800101	2
10050	800601	1
10050	820508	NULL
10050	830415	1
10030	800101	2
10030	820701	3
10030	830508	2
10080	810101	3
10025	800101	4
10090	800101	4

EMP3(EMPNO,TIME,JOBCLS)

EMPNO	TIME	JOBCLS
10010	800101	4
10010	810201	3
10010	821015	2
10005	810210	3
10050	800601	3
10050	820508	NULL
10050	830415	2
10030	800101	3
10030	820101	2
10030	830304	1
10080	810101	2
10025	800101	1
10090	800101	4
10090	811015	3

Table 3-6: The Relations Representing the VAs in EMP

EMP(ATTRIBUTE,PTYPE,LTYPE)

ATTRIBUTE	PTYPE	LTYPE
EMPNO	I4	1
NAME	C20	2
SEX	C1	2
DEPTNO	I2	3
JOBCLS	I2	3

Table 3-7: The Relation Describing the Scheme of EMP

which these VAs appear in the descriptive relation. The first VA (appearing after the last CA) is represented by REL_2, the next VA is represented by the relation REL_3, and so on. Using this convention, we name, for example, the regular relations underlying the TOR EMP: EMP1, EMP2 and EMP3, and the meta-data relation EMP.

The internal views of all the TORs in our TODB, as well as their representing relations, are included in Appendix A. The time slices presented so far in this dissertation for some TORs, are based on this information.

The sequence of relations, EMP, EMP1, EMP2 and EMP3, as presented above, together with the interpolation functions for the various VAs, provide the full information needed to construct the whole cube. All the interpolation functions for the VAs included in our benchmark database are step functions, identical to the one used earlier in this chapter as an example. Let us present its mathematical definition: if T_1 is the first time-point for which there is a value for a specific VA A_i in some TOR, and T_2 is the last time-

point having the same property, than the value of A_i in this TOR at time-point t, denoted by $A_i(t)$, is:

1. NULL, if $t < T_1$
2. $A_i(T_2)$, if $t > T_2$
3. $A_i(T)$, where T is the largest time included in the partial specification such that $T \leq t$, for any t such that $T_1 \leq t \leq T_2$

Basically, the interpolation functions should be supplied by the DBA, and there are many ways it can be done. The TDMS calls these functions whenever needed, assuming they have been supplied by the user as a part of the LINK step in creating the executable file for the TDMS.

One final comment should be made concerning the NULL values. Conceptually, NULL values are included in the cube for all VAs of a specific object at all time points before the first one at which there are explicitly recorded non-NULL values. In the representing relation, however, there is no need to record any NULL value at any time point before the first one at which some value is recorded, since the values at all these time points are interpreted as NULL by the interpolation function. NULL values should be recorded, whenever appropriate, only at time points after the first one for which a non-NULL value has been recorded (e.g., for the VAs of an employee who quits his job).

One can identify an immediate advantage of data structures based on the temporal differentiation of attributes, namely the resulting data structures contain a separate relation to represent each VA in a TOR (and one relation

representing all its CAs). In terms of storage space, it appears to be generally more efficient than the methods used in other research efforts in designing TDMSs, like [Snodgrass 84] and [Lum 84]. In those works, the time stamps are associated with the whole tuple. Therefore, if the value of one attribute in some tuple changes, the entire tuple has to be re-recorded with the new time stamp, even though only one value has changed. It is a reasonable assumption that the VAs in a TOR do not necessarily change simultaneously, and therefore our method would require less memory space.

3.5. Creating the TODB

Knowing the data structures representing a TOR in memory, we can now describe the ways to define TORs, load them with data and update them. This section covers these issues, including the procedures needed to maintain the TODB's consistency.

3.5.1. Defining a TOR scheme

In creating TORs, our TDMS has to map each TOR to a sequence of regular relations. The process entails the specification of:

* The TOR name
* The CAs
* The VAs
* The physical type (integer, real or character) of each CA and each VA.

The TDMS, then, uses this information to create all relations needed to represent this TOR. In addition, the system maintains a general table,

describing the overall TODB scheme. For our benchmark database, this information is included in Table 3-8.

MAINTB(TORNAM,CRDATE,TYPE)

TORNAM	CRDATE	TYPE
EMP	800101	0
SAL	800101	0
DEPT	800101	0
COURSE	800101	0
TRNHST	800101	0
EMPSAL	800811	1
EMPDPT	801023	1
EMPTRN	810205	1
.....	.....	....
.....	.....	....
.....	.....	....

Table 3-8: The MAINTB Relation of the TODB

This table, called MAINTB, contains the name of each TOR, its creation date and its type. The type is 0 for TORs created originally by the user (base TORs), and 1 for TORs created by the system in response to a query (derived TORs). This distinction serves to allow the user to load data only into TORs with type 0. MAINTB is the first item of information the system uses in order to perform any operation on the TODB.

3.5.2. Appending Data to a TOR

After its definition, the loading of the TODB with data is a fairly straightforward task. According to the concept of TODBs, no modification of already stored data is allowed, but rather data is constantly being appended.

In appending data, the necessary operations to maintain the database consistency are carried out. Basically, these operations are as follows:

* In adding a new object to a TOR, the TDMS verifies that such an object does not yet exist. To do this, only the relation containing the CAs of this TOR has to be accessed and checked; no other relations in this TOR need be examined. Whenever an object is added to a TOR, values (other than NULL) should be assigned to all its CAs. From this point on, no changes can be made to the CAs of this object.

* After adding a new object, there is no need to assign any values to any of its VAs. They are automatically interpreted as NULL for all time points, as long as values are not assigned to them. This means, according to our design, that the object does not exist in the database (yet), but once a non-NULL value is assigned to one of its VAs, the object starts to exist.

* In order to append a value to a VA of an existing object, the key is used to identify the object. First, the TDMS verifies that such an object really exists in the TODB. As before, only the relation containing the CAs of this TOR is involved in this verification. If the object does not exist, any attempt to append data to any of its VAs is rejected. If the object exists, then a <time-point, value> pair should be supplied in appending a value to any of its VAs. In executing the append operation, the TDMS maintains the finality property [Ariav 83c], by verifying that this object does not yet have an explicit value (a tuple) for this VA, at this time point. Only the relation containing this VA is involved in this operation. Recall that the key of this relation is the key of the TOR and the attribute TIME; since the user has supplied both, the system can perform this check. If such a tuple is found, the transaction is rejected, Otherwise, a new tuple, containing these values of the key attributes, the time value and the value of the VA, is appended to the relation.

Under the temporal differentiation of attributes, verifying the existence of an object requires accessing only the CAs relation, and the maintenance of the finality property requires accessing only the relation containing the data of the VA involved. Therefore, consistency is maintained efficiently, and the involvement of relations in the integrity checks is limited to those directly affected by the update.

The activities detailed above constitute the entire mechanism for consistency maintenance of the original TORs. However the database is expected to contain also views created by temporal relational algebra operation, containing information derived from base TORs. We assume that only base TORs can be loaded with data, avoiding thereby the complicated issues of updating derived views.

3.6. Summary

This chapter presented the data structures designed to efficiently store the information needed to construct the whole cube. The information stored is indeed minimal, since it contains only the constant values of the CAs, and the values of the VAs for the time points in which changes have taken place. Furthermore dropping of any item of data from the amount stored, clearly leads to loss of information. In using regular relations, we take full advantage of the temporal differentiation of attributes, by allocating a separate relation to each of the VAs, and another relation to all the CAs (whose temporal variation is, of course, the same).

The method of using regular relations does not require any additional

indexing schemes to capture the full meaning of the data (e.g., [Lum 84]), and allows for a relatively simple procedure to load data into the various relations representing the TOR. The use of regular relations carries an additional significant advantage. It releases us from building the tools to manipulate the basic data structures, since the regular relations can be handled by an existing DBMS.

Chapter 4

Introduction to the Temporal Relational Algebra Operations

4.1. Introduction

In the previous chapter, we described the data structures through which the cubic conceptual view of a TOR is represented. In the next three chapters, we deal with the definitions and the design of the temporal relational algebra operations which manipulate these data structures.

In dealing with the temporal operations, we define correspondence between the user's view of these operations and their implementation as executed by the TDMS. Conceptually, new TORs are created by these operations, but at the implementation level, these new TORs are represented by regular relations that are created by operations on similar relations representing the original operands.

4.2. Conceptual Definition of the Temporal Operations

Chapters 5, 6 and 7 cover the temporal relational algebra operations in full detail. Chapter 5 deals with the design of the temporal SELECT operation which is a natural extension to the regular SELECT operation. However, due to the temporal nature of this operation, there are actually two variations of it, the SELECT SOMEWHEN operation and the SELECT EVERYWHEN operation. In

addition, this chapter defines a temporally-oriented operation, the time selection. Chapter 6 analyzes the temporal PROJECT operation. In addition, it covers another temporally-oriented operation, the time projection, that is applicable in PROJECT operations that do not preserve the entire key. Chapter 7 analyzes the temporal JOIN operation. This current chapter outlines the general framework for designing and analyzing these operations.

The temporal SELECT, PROJECT and JOIN operations are defined as direct extensions to the corresponding regular relational operations. Being temporal operations, their semantic base is the external cubic view, on which they are defined. However, in their specific definitions, as well as in the analysis of their results, we use the definition of the cube as consisting of all its time slices, to make the extension from a regular operation to the corresponding temporal operation. As discussed in Chapter 3, the cube consists of a sequence of time slices, each of which corresponds to a particular time point, and is a valid regular relation containing the data of this specific time point. As such, this time slice may participate in any regular relational algebra operation, which is well defined in the regular relational model [Ullman 80], [Maier 82]. This observation is the basis of our definition of a correctness criterion for temporal operations, as follows:

> Given a regular relational algebra operation S (either a SELECT or a PROJECT), a temporal operation S' is its natural extension if, when operating on a TOR (cube) T, it produces a new TOR (cube) N whose various time slices are the results of the operation S, operating on the various time slices of this original TOR T.
>
> Formally, if the original TOR T is the union of the time slices (relations): T_1, T_2, ..., T_i, ..., T_n, and that the new TOR is the union of the time slices: N_1, N_2, ..., N_i, ..., N_n, then, the temporal

operation S', such that N = S'(T) (i.e., operation S' applied to T) is a natural extension of the regular operation S, if for every i (between 1 and n): $N_i = S(T_i)$

This definition implies that a temporal operation should be defined in correspondence with the way the cube is conceptually constructed from its various time slices. More specifically, conceptually the operation is executed through a loop on the time slices of the operand. In each step of this loop, the corresponding regular operation is executed on one time slice of the operand, producing the corresponding time slice of the resulting cube. The union of these new time slices constitutes the new cube.

This definition applies to the temporal SELECT and PROJECT operations. For the temporal JOIN operation, being a binary operation, the definition is slightly modified, as follows:

Assume that the TOR (cube) R is the union of the time slices: R_1, R_2, ..., R_n, and that the TOR (cube) S is the union of the time slices: S_1, S_2, ..., S_n. Then, a TOR T, the union of the time slices: T_1, T_2, ..., T_n, is the result of their temporal JOIN' operation, T=JOIN'[R,S], iff for every i (between 1 and n): $T_i = JOIN[R_i,S_i]$ (i.e., JOIN' extends JOIN).

Again, the resulting TOR is conceptually constructed by looping through all the time slices of the operands, corresponding to the same time points. In each step of the loop, a regular JOIN is executed with these two time slices, resulting in a new relation. The new TOR is the union of all these new relations.

One comment should be made about the use of the same n as the number of time slices included in the two operands. It does not imply anything about

the earliest time point or the latest time point at which information is recorded in the two operands. Any cube can conceptually be augmented both backward and forward in time. Time slices can conceptually be added before the earliest point at which information is explicitly recorded, by copying the CAs of all the objects in the cube, and inserting NULL values for all their VAs. Similarly, time slices can be appended after the last time point at which information is explicitly recorded, by simply copying the entire previous time slice (that is valid anyway, as long as no other information is recorded after this last time point). This observation implies that the two operands can be brought to consist of the same number of time slices.

The motivation to define a temporal operation as a repetitive execution of the corresponding regular operation on the various time slices of the operand(s), comes from the view of a TODB as a union of all the static databases that could have been created for the same relational schemes at all time points, starting at some time point t_0. In the TODB, we have all these possible databases included in one historical database. A typical TODB contains TORs originally created by the user (base TORs), and TORs resulting from temporal operation (views). Conceptually, a user should be able to use the TODB in order to create a static relational database, corresponding to a particular time point. Such a database would contain relations derived from base TORs, and relations derived from views. Each relation derived from a view should be the result of the regular operation on the relation(s) derived from the base TOR(s), corresponding to the temporal operation that created this view in the TODB. Such a situation maintains consistency in the conceptual

view of the TODB, as a union of static databases, with respect to both base TORs and views.

An alternative criterion for the correctness of a temporal operation might state that such an operation is correct if whenever it is applied to a cube that happens to be a single time slice (that is practically a regular relation), it produces a new time slice identical to the relation that would have been produced by the corresponding regular operation on the original time slice. The formal definition of this criterion is:

The temporal operation S' (an extension of the regular operation S) satisfies this criterion if:

$$S'(R) = S(X)$$

where R is a cube containing only one time slice, and X is its snapshot representation.[1]

We consider this criterion to be weaker than the previous one, since it does not address the entire cube. The first criterion is therefore stronger, since it treats the most general case (a general cube; not a single time slice). A temporal operation that satisfies the first correctness criterion, automatically satisfies the second, but not vice versa.

---

[1] a snapshot representation of a time slice is the presentation of its content in the format of a regular relation

4.3. The Framework of Analyzing a Temporal Operation

In presenting the temporal relational algebra operations included in our model, we use a syntax that has not been used in any previous research. It is, however, somewhat similar to the retrieval language TOSQL [Ariav 85], a temporal extension of a subset of SQL [Astrahan 75], containing the necessary extensions for the specifications of the query's time aspects. The definition of the temporal operations provides the basis for understanding their semantics, but does not outline the way to implement them. The definition of a temporal operation implies the execution of the same operation on the various time slices of the operand(s). Our implementation, however, uses the data structures representing the cube(s), and manipulates them to produce another sequence of the same data structures, representing the new cube. The procedure according to which the new data structures are created does not concern the user, who deals with the external cubic view of the operands, and with the same view of the resulting TOR.

Our correctness criteria serve as the basis for examining the correctness of a temporal operation definition. The strong correctness criterion implies that every time slice of the resultant TOR, at an arbitrary time point, should be equivalent to the result of the regular relational operation executed on the corresponding time slice(s) of the operand(s). The weak correctness criterion implies that a temporal operation, executed on a TOR (or TORs) that happens to contain only a single time slice, will produce an equivalent result to the one produced by the corresponding regular operation. The two correctness criteria are independent of the data structures used to represent

the TORs in the system, and of the way they are manipulated in the procedure that creates the new TOR.

It should be noted here, that the two time slices (the one taken from the resulting TOR, and the other produced by the regular operation on the corresponding time slice(s) of the operand(s), should be equivalent to satisfy either correctness criterion, but not necessarily identical. The definition presented in Chapter 3, implies that tuples in either time slices, containing NULL values for all their VAs, should be ignored, and if the remaining tuples are respectively identical, then the two time slices (relations) are equivalent. This definition is used to demonstrate that the operation satisfies a correctness criterion.

In the detailed analysis of the temporal operations, included in Chapters 5, 6 and 7, an attempt is made for each operation to apply the strong correctness criterion. The weak criterion is used only when the strong one fails. There are some operations that do not satisfy the strong criterion, but they are still meaningful and produce valuable information. For such operations, attempts are made to demonstrate that they satisfy at least the weak criterion. The next three chapters deal with the temporal relational algebra operations. All the operations are presented and analyzed using examples taken from our benchmark database. Their semantics are explained both for the specific examples and in general terms. Then, an algorithm is presented for each example, detailing the manipulation of the relations that represent the operand(s) which create the new relations representing the resulting TOR. These procedures are presented both in general terms, and

specifically for each example. Finally, each discussion ends with a verification step, examining and demonstrating the operation's correctness. The examples in the following three chapters are all based on our benchmark TODB, which is presented in Appendix A.

4.4. Summary

This chapter outlined the principles and the framework for the discussions about the temporal relational algebra operations. It defined the notion of "natural extensions" to the regular operations, and provided ways to demonstrate the correctness of operations against these definitions.

Chapter 5

The Temporal SELECT Operation

This chapter is the first of three in which we discuss the temporal operations. It covers the time-slice operation and its generalization (slicing the information of more than one time point) and the temporal SELECT operation. The first operation is, in fact, a special case of the second, and both are not extensions to regular relational operations, but rather new temporal operations unique to TODBs.

5.1. The Time-Slice Operation

The time-slice operation creates a new TOR, containing the data of exactly one time point (in our set of examples: one day), from the complete historical data stored in a specific TOR. It is explained using QUERY 5.1. The syntax for such a query is:

QUERY 5.1

```
CREATE TIME-SLICE
FROM EMP
AT 811020
INTO EMPQA
```

The definition of the time slice operation at the external level is as follows:

Given a TOR T, where T is the union of n flat relations R_i:

$$T = \text{UNION}_{i=1}^{n}(R_i),$$

then, the time slice from T at t = $\begin{cases} R_t \text{ for } 1 \leq t \leq n \\ R_n \text{ for } t > n \end{cases}$

QUERY 5.1 creates a new TOR containing only the data of 811020. In the user's external view, the result is a new cube containing the data of a single day. The relations representing the resulting TOR are included in Table 5-1. These new relations are produced by manipulating the relations representing the original TOR EMP, as included in Appendix A.

EMPQA(ATTRIBUTE,PTYPE,LTYPE)

ATTRIBUTE	PTYPE	LTYPE
EMPNO	I4	1
NAME	C20	2
SEX	C1	2
DEPTNO	I2	3
JOBCLS	I2	3

EMPQA1(EMPNO,NAME,SEX)

EMPNO	NAME	SEX
10010	MIKE	M
10005	MARY	F
10050	DAVID	M
10030	HENRY	M
10080	ALICE	F
10025	OSCAR	M
10090	SUSAN	F

EMPQA2(EMPNO,TIME,DEPTNO)

EMPNO	TIME	DEPTNO
10010	811020	2
10030	811020	2
10090	811020	4
10025	811020	4
10005	811020	2
10050	811020	1
10080	811020	3

EMPQA3(EMPNO,TIME,JOBCLS)

EMPNO	TIME	JOBCLS
10010	811020	3
10030	811020	3
10090	811020	3
10025	811020	1
10005	811020	3
10050	811020	3
10080	811020	2

Table 5-1: The Relations Representing the Time Slice from EMP at 811020

The resulting TOR contains the data of a single day. It is, however, a special case of a cube and a perfectly valid TOR.

In order to present the time-slice algorithm, let us assume that we are creating a time-slice RELQA from the TOR REL at a time point t. If REL has n VAs, then its descriptive relation is REL, its CAs relation is REL_1, and the relations representing its VAs are: REL_2, REL_3, ..., REL_{n+1}. Consequently, the new TOR RELQA will be represented by the relations: RELQA, $RELQA_1$, $RELQA_2$, ..., $RELQA_{n+1}$. The construction of the new relations is achieved through the following Algorithm 5.1:

1. Copy the relation REL to RELQA (EMP to EMPQA in our case), since the two TORs have the same definition.

2. REL_1 and $RELQA_1$ (EMP1 and EMPQA1 in our case) represent the CAs in the two TORs respectively. Therefore, they are identical, and REL_1 should simply be copied to $RELQA_1$. While copying it, sort it with its key, for the sake of efficiency in executing the following steps.

3. For each of the relations representing the VAs, perform the following procedure (explained for one of them, say REL_j):

   a. Sort the relation REL_j with its natural key (the key of the TOR itself and TIME).

   b. For each object included in $RELQA_1$, if there is a value in REL_j for TIME=t (TIME=811020 in our case), then copy it to $RELQA_j$. Otherwise, determine the value of this VA in this object at TIME=t by interpolation, using the available values in REL_j and the interpolation function, and record this value in $RELQA_j$ together with this object's key and with the value TIME=t. (Note that this procedure covers also the case in which the time point t is before the first time point appearing in REL_j for a specific object, or after the last time point for which this object has an explicit value in this relation, because of our definition of the interpolation function.) Generally, in any case of creating a new TOR, the association of interpolation functions for its VAs is inherited from the original operand(s).

Table 5-2 presents the "snapshot presentation" of this time slice. In this particular case, Table 5-2 is not one of the many snapshots constructing the whole cube; it is the whole cube. QUERY 5.1 takes only one time slice from the original TOR. Therefore, in this specific case the resulting TOR is actually reduced to a regular (non-temporal) relation.

The new TOR EMPQA is a subset of the TOR EMP, containing information at a single time point. However, if one tries to use this TOR in order to get

The Key	The CAs		The VAs	
EMPNO	NAME	SEX	DEPTNO	JOBCLS
10010	MIKE	M	2	3
10005	MARY	F	2	3
10050	DAVID	M	1	3
10030	HENRY	M	2	3
10080	ALICE	F	3	2
10025	OSCAR	M	4	1
10090	SUSAN	F	4	3

Table 5-2: A snapshot of the Result of QUERY 5.1 at 811020

information about other time-points, the results are semantically wrong, and obviously differ from those obtained using the original TOR EMP. The responsibility to avoid such a misuse of a TOR is necessarily the user's. He has to recognize that the only proper use of such a TOR is for the period for which it was created, for instance: the single day 811020 in the current example. This potential misinterpretation (i.e., making an inference about an object's status at 811022) is an error which is not unique for TDMSs, and could be committed in any information system.

The potential misuse of a derived TOR, with respect to the temporal dimension, could be controlled by the system, by associating two time values with each attribute in each TOR, indicating the earliest and the latest time-points for which it is valid. Such a mechanism should be studied further before adopting it.

5.2. The General Time-Selection

The general time-selection creates a new TOR containing the information not only of one time point, but of a whole time interval. It is actually a generalization of the previous operation. Conceptually, the result of a time selection operation consists of all the time slices corresponding to the various time points included in the period for which it is derived. Therefore, this new TOR is a cube that is "closed" by the lower limit and by the upper limit of the time-interval for which it is created.

The definition of the time selection operation at the external cubic level is as follows:

Given a TOR T, where T is the union of n flat relations:

$$T = \text{UNION}_{i=1}^{n}(R_i),$$

where R_i is the time slice of time point i.

then, the time selection of a TOR T during the interval [k,l], is defined as follows:

$$\text{TIME-SELECTION}(T)[k,l] = \text{UNION}_{i=k}^{l}(R_i)$$

In order to define this operation in terms of the internal view, let us use the following definition of the internal view, as a union of its various objects:

Let OB_i be the representation of the object i in terms of its CAs and the partial specifications (including the interpolation functions) of its VAs (i.e., it contains a single value for each of this object's CAs and a function from time points to values for each of its VAs). Then, the internal view of a TOR that contains n objects is:

$$V = \text{UNION}_{i=1}^{n}(OB_i)$$

then, the definition of the time selection operation in terms of the internal view is:

$$\text{TIME SELECTION}(V)[k,l] = \text{UNION}_{i=1}^{N}(OB_i[k,l])$$

where $OB_i[k,l]$ is the data of the object i, that corresponds to the interval [k,l].

Let us present the creation of the general time-selection, through QUERY 5.2. This query conceptually selects out from the TOR EMP all its time slices during the period 830101 - 831231, and organizes these time slices in the new cube EMPQB.

QUERY 5.2

```
CREATE TIME SELECTION
FROM EMP
DURING 830101 - 831231
INTO EMPQB
```

The implementation of the time selection operation, in terms of the representing relations, is presented for a general TOR REL, represented by the relations REL, REL_1, REL_2, REL_3, ..., REL_{n+1}. The time selection operation creates the new TOR RELQB, containing the time slices between the time points t_1 and t_2. This new TOR is represented by the relations: RELQB, $RELQB_1$, $RELQB_2$, ..., $RELQB_{n+1}$. The procedure to create these new relations is an extension of the algorithm described for QUERY 5.1, and is included in the following Algorithm 5.2:

1. Copy the relation REL to the new relation RELQB, since the two TORs have the same definition.

2. Copy the relation REL_1 representing the CAs in the TOR REL to the relation $RELQB_1$ representing the CAs in RELQB, since the CAs are not affected by the time selection. Also, sort this relation for the sake of efficiency in executing the following steps.

3. For each of the relations representing the VAs, execute the following steps (described for one of them, say REL_j):

   a. Copy all tuples in REL_j with $t_1 \leq TIME \leq t_2$ to the relation $RELQB_j$.

   b. For each object that does not have a value in REL_j for $TIME=t_1$, create a value in $RELQB_j$ for this time point by interpolation, using the latest value recorded for this object in REL_j before t_1, and the earliest value recorded in it after t_1. The interpolation will, of course, properly handle the cases in which one of these values, (possibly both) does not exist.

   c. For time points after the upper limit of the time interval for which it has been created, insert NULL values in $RELQB_j$ for all objects at $TIME=t_2+1$. This step "closes" the new cube at the upper limit of its time interval. This step serves to prevent inferences about a time point outside the range $[t_1,t_2]$.

Step 3_b in this algorithm enables the correct derivation of all the VAs' values for all the objects of the TOR at any time point between t_1 and t_2. Step 3_c is aimed to produce a result in which all the values of all the VAs for all the objects will be NULL, in any use of the resulting TOR at a time point outside its "legal" time-interval. These results should alert the user to recognize this "misuse" of the TOR. The same solution could have been adopted for the time slice operation, to solve the same problem. Therefore, let us assume that even for a time slice, which is a special case of the general time selection, applied to a single day, Algorithm 5.1 is replaced by Algorithm 5.2, that contains this "closing" step.

The relations representing EMPQB are included in Table 5-3.

EMPQB(ATTRIBUTE,PTYPE,LTYPE)

ATTRIBUTE	PTYPE	LTYPE
EMPNO	I4	1
NAME	C20	2
SEX	C1	2
DEPTNO	I2	3
JOBCLS	I2	3

EMPQB1(EMPNO,NAME,SEX)

EMPNO	NAME	SEX
10010	MIKE	M
10005	MARY	F
10050	DAVID	M
10030	HENRY	M
10080	ALICE	F
10025	OSCAR	M
10090	SUSAN	F

EMPQB2(EMPNO,TIME,DEPTNO)

EMPNO	TIME	DEPTNO
10010	830101	2
10010	840101	NULL
10005	830101	2
10005	840101	NULL
10050	830101	NULL
10050	830415	1
10050	840101	NULL
10030	830101	3
10030	830508	2
10030	840101	NULL
10080	830101	3
10080	840101	NULL
10025	830101	4
10025	840101	NULL
10090	830101	4
10090	840101	NULL

EMPQB3(EMPNO,TIME,JOBCLS)

EMPNO	TIME	JOBCLS
10010	830101	2
10010	840101	NULL
10005	830101	3
10005	840101	NULL
10050	830101	NULL
10050	830415	2
10050	840101	NULL
10030	830101	2
10030	830304	1
10030	840101	NULL
10080	830101	2
10080	840101	NULL
10025	830101	1
10025	840101	NULL
10090	830101	3
10090	840101	NULL

Table 5-3: The Relations Representing the Result of QUERY 5.2

Notes:

1. Some values of DEPTNO and JOBCLS for TIME=830101 in EMP2 and in EMP3 are missing. Therefore, for all objects that do not have a value either in EMP2 or in EMP3 for 830101, we infer the "missing" values by interpolation, and record them in EMPQB2 and EMPQB3 respectively.

2. NULL values are inserted in both EMPQB2 and EMPQB3 for all objects at 840101. This step "closes" the cube for inquiries of the new TOR for a time point after 831231.

The operations discussed so far in this chapter highlight one of the advantages of the temporal differentiation of attributes, namely that it suggests a way to decompose temporal operations into a sequence of "localized" operations on regular relations.

5.3. The SELECT SOMEWHEN Operation

The temporal extension of The relational SELECT operation presents conceptual problems with respect to their meaning. Basically, the SELECT operation, as noted in Chapter 4 in [Ariav 83a] and in [Clifford 85a], is divided into two categories: SELECT WHERE <selection-expression> SOMEWHEN, and SELECT WHERE <selection-expression> EVERYWHEN. The difference between these two operations is that the SELECT SOMEWHEN should select all objects with tuples that meet the selection criterion, without considering other tuples of the same objects. The SELECT EVERYWHEN operation is concerned with the entire history of an object, and selects only those objects whose all tuples satisfy the selection criterion, without exception. QUERY 5.3 demonstrates the SELECT SOMEWHEN operation.

QUERY 5.3

```
SELECT FROM EMP
WHERE DEPTNO = 2 SOMEWHEN
INTO EMPQC
```

SELECT SOMEWHEN cannot be resolved on a tuple basis, but rather on an object-level, namely, that all the information of the relevant objects is recorded in the new TOR. This approach puts a slightly different interpretation on the SELECT SOMEWHEN operation, compared to the regular SELECT, as it introduces to the resulting TOR tuples that can be considered as irrelevant, since they do not satisfy the predicate.

[Clifford 85b] points out, however, that this is, in fact, the correct interpretation of the temporal SELECT. He argues that each of the temporal SELECT, PROJECT and time selection operations should exhibit "dimensional purity", i.e., SELECT on objects, PROJECT on attributes, TIME-SELECTION on time. TOSQL in [Ariav 83a] is designed in that way.

According to this approach to the SELECT SOMEWHEN, all the tuples of those objects that have some qualifying tuple(s) are recorded in the resulting TOR. In our example, it implies that all the information of the relevant employees will be copied from EMP to EMPQC without any change, thus avoiding the need to distinguish between relevant periods and irrelevant ones. In order to understand the logic underlying this approach, one can think of the same database handled manually. If all files of employees who worked at some

period of time for department 2 are selected, then this operation does not change the information in those files for other periods of time.

The SELECT SOMEWHEN operation does not satisfy the strong criterion of correctness, suggested in Chapter 4. Potentially, tuples, that in themselves are irrelevant, are included in the resulting TOR, as they belong to objects that have some qualifying tuple(s). This operation, however, satisfies the weak criterion of correctness, requiring that the result of this operation on a single time slice produces the same result obtained by the regular operation. This criterion is satisfied because each object in such an operand contains only the information of one single time point, included in its only tuple in this TOR. Therefore, if this tuple qualifies, then it is included in the resulting TOR, but no other tuples of the same object are included, simply because there are no such tuples.

The SELECT SOMEWHEN operation can be properly defined on the internal view, as this view consists of the various objects in the TOR. The SELECT SOMEWHEN does not create new objects, but selects old objects of the operand according to the selection criterion.

Let the internal view of the TOR be (see formal definition earlier in this chapter):

$$V = \text{UNION}_{i=1}^{n}(OB_i)$$

The definition of the SELECT SOMEWHEN operation, in terms of internal views is:

$$\text{SELECT } [V] \text{ WHERE}(C) \text{ SOMEWHEN} = \text{UNION}_{i=1}^{k}[OB_i \text{ WHERE}(C)]$$

where [OB_i WHERE (C)] is a qualifying object, namely an object that has at least one qualifying tuple in the TOR.

The algorithm for implementing the SELECT SOMEWHEN operation is designed for the general case of a TOR REL, containing n VAs, and represented by the relations: REL, REL_1, REL_2, ..., REL_{n+1}. A new TOR RELQC is to be created as a result of a SELECT SOMEWHEN operation, in which the WHERE clause may be any predicate defined in [Ariav 83a]. This new TOR will be represented by the relations RELQC, $RELQC_1$, $RELQC_2$, ..., $RELQC_{n+1}$, and is created through the following Algorithm 5.3:

1. Copy the relation REL into the new relation RELQC, since the two TORs have the same definition.

2. Scan all the representing relations of REL, that are involved in the WHERE clause, and prepare a set of all the objects that have at least one tuple that qualifies according to this predicate. All these objects will be included in the new TOR RELQC.

3. Copy, from each relation REL_i, <u>all</u> the tuples belonging to the objects in this set, to the corresponding relation $RELQC_i$.

This operation, as applied to QUERY 5.3, will be carried out through the following steps:

1. Copy the regular relation EMP into the regular relation EMPQC, since the new TOR has the same scheme as the TOR EMP.

2. Scan the relation EMP2, and prepare a list of keys (EMPNOs) that has the value DEPTNO=2 at least once in this relation.

3. Copy <u>all</u> the tuples with these keys from EMP1, EMP2, and EMP3 to EMPQC1, EMPQC2 and EMPQC3, respectively.

Let us conclude this example by presenting the internal view of EMPQC, in

Table 5-4. A typical snapshot (at the arbitrary day 820701) from the resulting TOR is presented in Table 5-5.

EMPNO	NAME	SEX	DEPTNO	JOBCLS
10010	MIKE	M	800101 3 810215 2	800101 4 810201 3 821015 2
10005	MARY	F	810210 2	810210 3
10030	HENRY	M	800101 2 820701 3 830701 2	800101 3 820101 2 830304 1

Table 5-4: The Internal View Describing the Result of QUERY 5.3

The Key	The CAs		The VAs	
EMPNO	NAME	SEX	DEPTNO	JOBCLS
10010	MIKE	M	2	3
10005	MARY	F	2	3
10030	HENRY	M	3	2

Table 5-5: A Snapshot from the Result of QUERY 5.3 at 820701

Note that Table 5-5 contains employee 10030, even though his DEPTNO at 820701 is 3, namely his tuple for this day does not satisfy the selection criterion. Regardless, this tuple is included in the resulting TOR, since this employee has other tuples that do satisfy the selection criterion. This is an illustration of the different interpretation of the SELECT SOMEWHEN operation, compared to the regular SELECT.

As mentioned above, this operation does not satisfy the strong correctness criterion. To illustrate this, let us take a snapshot from the original TOR EMP at 820701, and then select from this snapshot those tuples that satisfy the predicate DEPTNO=2. This results in a relation that is not equivalent to Table 5-5, since it does not contain the tuple of employee 10030, that is included in Table 5-5. Nevertheless, the operation satisfies the weak correctness criterion that deals only with a single time slice, since in such a case only qualifying tuples are selected.

QUERY 5.3 presents a SELECT SOMEWHEN operations with the WHERE clause, but without any time-predicate like AT, DURING, BEFORE, AFTER, and therefore the selection has been made for all time-points in the TOR. One may, of course, want to delineate a time period, and successively apply the SELECT SOMEWHEN operation and the time selection. QUERY 5.4 expresses this operation.

QUERY 5.4

```
SELECT FROM EMPQC
DURING 810501-820801
INTO EMPQD
```

Note that the operand, EMPQC, in query 5.4, is the result of QUERY 5.3. In other words, QUERY 5.4 is:

```
CREATE TIME SELECTION [ SELECT FROM EMP
                        WHERE DEPTNO = 2 SOMEWHEN ]
DURING 810501-820801
INTO EMPQD
```

QUERY 5.4 means to use the result of QUERY 5.3, which contains the entire history of employees who worked at some time point in department 2, and then further select from this TOR the tuples of the period 810501 - 820801. This query presents another problem, caused by the fact that the SELECT SOMEWHEN operation does not satisfy the strong correctness criterion. Consider the same two operations in reverse order: first, we create the time-selection, and then apply the SELECT SOMEWHEN operation to its result. Specifically, we first create the TOR EMPQE by:

```
SELECT FROM EMP
DURING 810501-820801
INTO EMPQE
```

And then, create the TOR EMPQF by:

```
SELECT FROM EMPQE
WHERE DEPTNO = 2
SOMEWHEN
INTO EMPQF
```

I.e., we evaluate

```
SELECT FROM  [ CREATE TIME SELECTION
             | FROM   EMP
             | DURING 810501-820801 ]

WHERE DEPTNO = 2 SOMEWHEN
INTO EMPQF
```

The resulting TOR EMPQF is not necessarily identical to the TOR EMPQE,

resulting from executing the same queries in reverse order. As an example for such a situation, assume that an employee worked for department 2, but not during the period 810501-820801. Then, the execution of QUERY 5.3 and then 5.4 will first pick out all his tuples from the TOR EMP into the TOR EMPQC, and then the time selection will have his tuples for the period 810501-820801, even though these tuples do not contain the value 2 for the attribute DEPTNO at all. In the reverse order, the time selection will not contain any tuple with DEPTNO=2 for this employee, and therefore the SELECT SOMEWHEN will ignore this employee completely.

This example demonstrates that time selection is not commutative with SELECT SOMEWHEN. The two cases, discussed above, represent different semantics: in the first one, we first wanted to select the complete histories of all employees who worked for department 2 in some period of time, and then to limit our interest in those employees to the period 810501-820801. In the second question, we first wanted to concentrate only on the data for the period 810501-820801, and then to select from that data the employees who worked for department 2 at some time points within the selected period. Therefore, the combination of the different semantics of the two questions, and our interpretation of the SELECT SOMEWHEN operation, leads to the result that these operations do not commute.

5.4. The SELECT EVERYWHEN Operation

The SELECT EVERYWHEN operation checks the objects in the existing TOR, and selects all the tuples of those objects that satisfy the selection predicate at all time points. This operation turns out to be much simpler than the SELECT SOMEWHEN operation; it does not exhibit any of the problems caused by the first operation, in which an object could have both qualifying and non qualifying tuples.

SELECT EVERYWHEN, like SELECT SOMEWHEN, can be properly defined in terms of the internal views, as follows:

$$\text{SELECT WHERE (C) EVERYWHEN} = \text{UNION}_{i=1}^{k}[OB_i \text{ WHERE (C)}]$$

where $[OB_i$ WHERE (C)] is a qualified object, namely an object whose all tuples qualify for this selection.

QUERY 5.5 demonstrates the SELECT EVERYWHEN operation. This query checks all the objects in EMP, and selects only those whose DEPTNO is 2 at all time points. Only one object qualifies: employee 10005, who joined department 2 at 810210, and never left it thereafter. Employees 10010 and 10030 worked for department 2 at some periods of time, but since they also worked for other departments at some periods of time, they do not qualify to be selected into the new TOR.

QUERY 5.5

```
SELECT FROM EMP
WHERE DEPTNO = 2  EVERYWHEN
INTO EMPQG
```

In order to describe the algorithm to execute the SELECT EVERYWHEN operation, let us define the general case. The TODB contains a TOR REL, containing n VAs, and represented by the relations: REL, REL_1, ..., REL_{n+1}. A new TOR RELQG is to be created by a SELECT EVERYWHEN operation, containing a WHERE clause, as defined in [Ariav 83a]. The resulting TOR will be represented by the relations RELQG, $RELQG_1$, ..., $RELQG_{n+1}$, created through the following Algorithm 5.4:

1. Copy the relation REL into RELQG, since the new TOR has the original TOR's definition.

2. Scan all the representing relations of REL, that are involved in the WHERE clause, and prepare a set of all the objects in REL whose all tuples qualify according to its predicate. This set contains the objects of the new TOR RELQG.

3. Copy, from each of the original relations REL_i, all the tuples belonging to the objects in this set, to the corresponding relation $RELQG_i$ in the new TOR.

The same procedure, applying to our specific example, QUERY 5.5, is as follows:

1. Copy the relation EMP to the new relation EMPQG, since the new TOR has the same scheme as the TOR EMP.

2. Scan the relation EMP2, and prepare a set of all objects (employees) who worked only for department 2.

3. Copy all the information about these employees from EMP1, EMP2 and EMP3 to EMPQG1, EMPQG2 and EMPQG3 respectively.

Table 5-6 presents the internal view of the new TOR EMPQG.

EMPNO	NAME	SEX	DEPTNO	JOBCLS
10005	MARY	F	810210 2	810210 3

Table 5-6: The Internal View Describing the Result of QUERY 5.5

Like the SELECT SOMEWHEN operation, the SELECT EVERYWWHERE is not commutative with the time selection operation. The reason for this is that time selection may isolate a period, where the required condition for an object prevails EVERYWHEN, while not being true elsewwhere. Applying the SELECT EVERYWHERE first, would have eliminated this object from the resulting TOR.

Since the SELECT EVERYWHEN is not commutative with the time selection operation, it cannot satisfy the strong correctness criterion. However, it does satisfy the weak criterion. If the operand is a single time slice, then the SELECT EVERYWHEN is identical to the SELECT SOMEWHEN operation, and therefore, like the SELECT SOMEWHEN, it satisfies the weak correctness criterion.

5.5. Summary

This concludes the discussion of the various select operations, the time selection, the SELECT SOMEWHEN operation and the SELECT EVERYWHEN operation. The time selection operation contains one well defined case, in which all data for a specific time interval is selected. Therefore, our example completely covers this operation. In the SELECT EVERYWHEN operation, an object is selected only if all its tuples qualify. In the SELECT SOMEWHEN operation an object is selected if it has at least one qualifying tuple.

Chapter 6

The Temporal PROJECT Operation

The PROJECT operation, as defined in Chapter 4 in [Ariav 83a] and in [Clifford 85a], conforms to the basic relational notion of projection as it manipulates the cube to create another TOR with a subset of the original TOR's attributes. The meaning of this operation is contingent upon the extent to which the original key attributes are retained in the result (i.e, all, some or none). This semantic problem relates to the loss of the original "object's identity" in the case of dropping some (or all) of the key attributes. As can be expected, the PROJECT operation presents further problems in TDMSs.

The conceptual definition of the temporal PROJECT operation, used throughout this chapter, follows the guidelines set in Chapter 4, as follows:

A TOR S is a result of a temporal PROJECT operation on the TOR R, if every time slice of S is the result of the same regular PROJECT on the corresponding time slice of the operand R. The definition of the temporal PROJECT' operation at the external level is:

$$\text{PROJECT}'(T)\ [R_1 \ldots R_m] = U_{i=1}^{n} \text{PROJECT}(R_i)[R_1 \ldots R_m]$$

Defining the temporal PROJECT operation in terms of the internal view is confusing: it is appropriate in cases where the operation preserves the key, but useless if the key is not preserved.

Our initial analysis indicated that the result of the projection is sensitive to the attributes included in it. To assure comprehensive study, we have identified all possible attribute combinations (see Table 6-1), and examined each one of them. Basically, two major types emerged. The first encapsulates cases 1 through 9 in Table 6-1, in which the key is entirely preserved. The second type captures the essence of cases 10 through 27, in which all or part of the key is deleted.

6.1. PROJECT Operations that Preserve the Key

This section covers the PROJECT operations in which all the key attributes are projected onto the new TOR. In this situation, the objects in the new TOR are exactly the ones in the old TOR; furthermore, the new TOR has the same meaning as the old one, except that it contains fewer attributes per object.

Case 1 in Table 6-1, in which the resulting TOR is identical to the source TOR, involves no manipulation of data, and therefore no further discussion is needed. QUERY 6.1 exemplifies case 2 in Table 6-1, and presents a simple PROJECT operation, in which the key is preserved, and only one attribute is dropped (a CA in our case).

QUERY 6.1

```
PROJECT EMP
ONTO EMPNO,SEX,DEPTNO,JOBCLS
INTO EMPQG
```

CASE	THE KEY	NON-KEY CAs	VAs
1	I	I	I
2	I	P	I
3	I	I	P
4	I	P	P
5	I	N	I
6	I	I	N
7	I	N	N
8	I	N	P
9	I	P	N
10	N	P	N
11	N	I	N
12	N	N	P
13	N	P	P
14	N	N	I
15	N	P	I
16	N	I	P
17	N	I	I
18	N	N	N
19	P	P	N
20	P	I	N
21	P	N	P
22	P	P	P
23	P	N	I
24	P	P	I
25	P	I	P
26	P	I	I
27	P	N	N

I means: attributes are included.
P means: attributes are partially included.
N means: attributes are not included.

Table 6-1: The Various Cases in the PROJECT Operation

QUERY 6.1 preserves the identities of the objects (employees) in the new TOR, and these objects just lose the attribute NAME in their transition from the old TOR EMP to the new TOR EMPQG.

In terms of the external cubic view, this query drops out the vertical slice containing the attribute NAME from the original TOR EMP. The new TOR contains the four remaining attributes. Each of its time slices is in itself a result of the same regular PROJECT on the corresponding time slice of the operand. Since the key is preserved, the original objects keep their identities in the new TOR, and none of the new TOR's time slices contains any duplicate tuples.

In order to describe the general steps to execute this PROJECT, assume that the original TOR is REL, containing m CAs CON_1, CON_2, ..., CON_m and n VAs, and is represented by the relations REL, REL_1, REL_2, ..., REL_{n+1}. The PROJECT operation creates a new TOR by dropping one CA, say the CA CON_k, from REL. The new TOR is RELQG, represented by the relations RELQG, $RELQG_1$, $RELQG_2$, ..., $RELQG_{n+1}$. The steps to create the new TOR RELQG are included in the following Algorithm 6.1:

1. Copy the descriptive relation REL to the relation RELQG which is the descriptive relation of the new TOR, except for the tuple that describes the attribute CON_k in REL. Each descriptive relation contains three attributes: ATTRIBUTE, PTYPE and LTYPE. This step is, therefore, achieved by the following regular SELECT operation:

   SELECT FROM REL
   WHERE ATTRIBUTE ≠ "CON_k"
   INTO RELQG

2. The attribute CON_k is stored in REL_1. Therefore perform the following regular PROJECT:

   PROJECT REL_1
   ONTO $CON_1,\ldots,CON_{k-1},CON_{k+1},\ldots,CON_m$
   INTO $RELQG_1$

3. Finally, copy the relations REL_2, REL_3, ..., REL_{n+1} to $RELQG_2$, $RELQG_3$, ..., $RELQG_{n+1}$ respectively.

The same steps, executed for our particular example, QUERY 6.1, are:

1. Create the descriptive relation, EMPQG, of the new TOR, by the following SELECT operation:

```
SELECT FROM EMP
WHERE ATTRIBUTE ≠ "NAME"
INTO EMPQG
```

2. The attribute NAME is stored in EMP1. Therefore, perform the following regular PROJECT:

```
PROJECT EMP1
ONTO EMPNO,SEX
INTO EMPQG1
```

3. Finally, copy EMP2 and EMP3 to EMPQG2 and EMPQG3 respectively.

Table 6-2 presents a typical snapshot (for the arbitrary day 811015) from the resulting TOR EMPQG.

The Key	A CA	The VAs	
EMPNO	SEX	DEPTNO	JOBCLS
10010	M	2	3
10005	F	2	3
10050	M	1	3
10030	M	2	3
10080	F	3	2
10025	M	4	1
10090	F	4	3

Table 6-2: A Snapshot from the Result of QUERY 6.1 at 811015

The snapshot presented in Table 6-2 is clearly identical to the result of a regular PROJECT operation on the snapshot taken from the TOR EMP at 811015.

The same is true for all other possible snapshots. This demonstrates that this temporal PROJECT satisfies the strong correctness criterion.

One advantage of the temporal differentiation of attributes is demonstrated here by the simplicity of the operation that has only to address the descriptive relation and the relation containing the attribute to be dropped, while copying all other representing relations without any changes.

Consider QUERY 6.2. In this query, corresponding to case 3 in Table 6-1, the VA JOBCLS is to be dropped from EMP.

QUERY 6.2

```
PROJECT EMP
ONTO EMPNO,NAME,SEX,DEPTNO
INTO EMPQH
```

Again, the resulting TOR has the same objects as EMP, with one attribute (a VA in this case) dropped.

Conceptually, there is no difference between QUERY 6.1 and QUERY 6.2, since in both of them the key is preserved. In their implementation, however, there is a difference, reflecting the difference between the types of the attributes being dropped in the two queries.

In the general case, the original TOR REL has n VAs: VAR_1, VAR_2, ..., VAR_n, and is represented by the relations: REL_1, REL_2, ..., REL_{n+1}. The PROJECT operation drops one VA, say VAR_k, and creates a new TOR RELQH, that

has only n-1 VAs, and is represented by the relations: $RELQH_1$, $RELQH_2$, ..., $RELQH_n$. The procedure to create the new TOR RELQH is included in the following Algorithm 6.2:

1. Execute the regular SELECT operation:

```
SELECT FROM REL
WHERE ATTRIBUTE ≠ "VAR_k"
INTO RELQH
```

2. Copy REL_1, REL_2, ..., REL_{k-1}, REL_{k+1}, ..., REL_{n+1} to $RELQH_1$, $RELQH_2$, ..., $RELQH_n$ respectively. All the relations, except REL_k, are copied without any change.

The same steps, applying to our specific example, QUERY 6.2, are:

1. Perform the regular operation:

```
SELECT FROM EMP
WHERE ATTRIBUTE ≠ "JOBCLS"
INTO EMPQH
```

2. Copy EMP1 and EMP2 to EMPQH1 and EMPQH2 respectively, and ignore EMP3.

Due to the conceptual similarity between QUERY 6.2 and QUERY 6.1, it is clear that the operation, in which a temporal PROJECT drops a VA, also satisfies the strong correctness criterion. In fact, this is true for any temporal PROJECT that preserves the key, since it maintains the objects' identities in all possible time slices, and consequently in the TOR itself.

The advantages of the temporal differentiation of attributes are even clearer in the evaluation of QUERY 6.2. As VAs are stored in separate relations, dropping such an attribute simply implies its deletion from the definition of the new TOR.

Consider QUERY 6.3, corresponding to case 4 in Table 6-1. As one CA and one VA are dropped, QUERY 6.3 is a combination of QUERIES 6.1 and 6.2, and is resolved accordingly. The same is true for cases 5 through 9 in Table 6-1.

QUERY 6.3

```
PROJECT EMP
ONTO EMPNO,SEX,JOBCLS
INTO EMPQI
```

6.2. PROJECT Operations That Do Not Preserve the Key

This section deals with PROJECT operations in which the whole key, or a part of it, is deleted, i.e., cases 10-27 in Table 6-1. From conceptual point of view, such PROJECT operations change the meaning of the TOR, and create new type of objects in the resulting TOR. From the implementation point of view, the projected attributes do not maintain their original types (e.g., key, CA, VA) in this new TOR. Consequently, the new TOR's key is not the same as the old TOR's, but, instead, it consists of all the projected attributes. The combination of these attributes defines the "objects" of the new TOR.

QUERY 6.4 corresponds to case 10 in Table 6-1, in which the key is not fully retained.

QUERY 6.4

```
PROJECT EMP
ONTO SEX
INTO EMPQJ
```

This query creates a new TOR as a result of a PROJECT operation on the TOR EMP, by dropping all its attributes, except the CA SEX. Consequently, the new TOR has a new kind of object, consisting only of the attribute SEX, and a new meaning, compared to the existing TOR. In this specific example, we know that there are only two potential objects in the resulting TOR: M and F. Either of these objects might not exist in the database at some points of time (e.g., if there was no woman in the company's personnel during the period 810510-810722, then the object F does not exist in the new TOR in this period), and the resulting TOR should be designed to reflect such situations. Since SEX is the only attribute in this TOR, it is obviously its key, and as such it exists in the database at all time points. Therefore, if the resulting TOR contains only the attribute SEX, then we are faced with the problem of how to indicate periods of time in which one of the objects (M or F) does not exist at all.

According to the general definition of temporal projections, every time slice of the resulting TOR is a result of the same regular PROJECT on the corresponding time slice of the operand. If such a PROJECT operation is applied to all the operand's time slices, it produces a sequence of regular relations, one per time-point, each of which contains either both of the objects M and F, or one of them, or none. Now, we conceptually combine them to form the new cube. The objects in this cube are, of course, M and F. However, they do not necessarily exist at all time points. Usually, such a situation is indicated by the assignment of NULL values to all the VAs of an object at the time-points in which it does not exist. In our case, there are no VAs to

play this role; therefore, without introducing another component to the resulting TOR, there is no way to combine the different relations, resulting from a PROJECT on the operand's time slices, to form a new TOR (cube).

A solution to this problem is in the introduction of another attribute, a VA, into the resulting TOR. This rather artificial attribute, called STATE, is aimed to allow the proper recording of the fact that objects of the new TOR may not exist at all time points. STATE will have NULL values for an object at all time points in which this object does not exist, following the convention of assigning NULL values to all VAs to indicate inexistence. At time points in which this object does exist, the VA STATE can have any non-NULL value, e.g., STATE=1. The procedure to create the new TOR determines the values of STATE for the various time points, based upon the data in the operand.

The VA STATE provides a technical solution to the problem caused by objects that exist at some periods of time, and do not exist in others. This problem, as illustrated in QUERY 6.4, can be analyzed conceptually from another angle, by considering the conceptual view of the new TOR, the cube. This new cube is created by dropping all the attributes of EMP except SEX. Now, let us consider an arbitrary time slice of the new cube, corresponding to some day. This time slice probably contains many values of both M and F, depending upon the number of men and women working for the company on this day. In the old TOR, these values belong to different objects. In the new TOR, however, multiple occurrences of Ms or Fs in one time slice are duplicates that should, of course, be deleted. This step of deleting

duplicates, leaves at most one M and one F in every time slice. However, as already indicated before, some time slices may not contain either the value M or the value F or both. The new VA STATE is needed to indicate such situations.

The new TOR EMPQJ, containing the result of QUERY 6.4, is represented by the relations included in Table 6-3.

EMPQJ(ATTRIBUTE,PTYPE,LTYPE)

ATTRIBUTE	PTYPE	LTYPE
SEX	C1	1
STATE	I2	3

EMPQJ1(SEX)

SEX
M
F

EMPQJ2(SEX,TIME,STATE)

SEX	TIME	STATE
M	800101	1
F	800101	1

Table 6-3: The Relations Representing the Result of QUERY 6.4

Any time slice taken from the new TOR (for a time point starting at 800101) contains both objects M and F. This reflects the fact that at any time point starting at 800101, both men and women were employed in this organization. Clearly, this is exactly the result of this PROJECT operation

on any of the original TOR's time slices. Thus, this PROJECT, like the ones covered earlier in this chapter, satisfies the strong correctness criterion.

Let us define QUERY 6.4 in general terms. The original TOR REL contains, among the rest, the non-key CA CONST, and is represented by the relations REL, REL_1, ..., REL_{n+1}. The new relation RELQJ is a result of a PROJECT operation, in which only the CA CONST is selected. The procedure to create the new TOR RELQJ is the following Algorithm 6.3:

1. Create the three relations: RELQJ(ATTRIBUTE,PTYPE,LTYPE), $EMPQJ_1$(CONST) and $RELQJ_2$(CONST,TIME,STATE).

2. Scan the relation REL_1, and identify the domain of CONST. These are the objects of the new TOR. Record them in $RELQJ_1$.

3. The following steps record the right values in the relation $EMPQJ_2$, representings the VA STATE. They are described for some object CONST=a in the new TOR.

   a. Scan the relation REL_1 again, and prepare the list of all the objects in the original TOR REL, having the value CONST=a in their tuples.

   b. With this list, scan all the relations representing the original TOR's VAs simultaneously, and locate the first time point at which some object in this list has a non NULL value at least in one of them (indicating that this object exists at this time point in the original TOR).

   c. If no such a time point is found at all, it means that the object CONST=a of the new TOR does not exist at all. Therefore, delete it from the relation $RELQJ_1$, and do not record anything for it in $RELQJ_2$. Then, proceed to the next object of the new TOR (if any).

   d. If such a time point, t, is found for some object of REL, say OBJREL, then record the tuple CONST=a, TIME=t and STATE=1 in $RELQJ_2$.

   e. Continue the scanning, looking for possible NULL values in all

the VAs of OBJREL at some time point. If no such a case is found, then nothing more has to be recorded for CONST=a in $RELQJ_2$.

f. If NULL values were found for all VAs of OBJREL at some time point t_1, then look for another possible object of REL having CONST=a, that exists at t_1. If no such an object is found, record the tuple CONST=a, TIME=t_1 and STATE=NULL in $RELQJ_2$. Now, keep scanning, looking for a possible object of REL, having CONST=a, that exists at a later time point. If found, denote this time point t, and return to step (d) above. If not, proceed to the next object of RELQJ (if any).

g. If NULL values were found for all the VAs of OBJREL at some time point, but another object of REL with CONST=a exists at this time point, then CONST=a still exists in RELQJ. Denote the new object by OBJREL, and return to step (e) above.

The same procedure, applied specifically to our QUERY 6.4, is as follows:

1. Create the relations EMPQJ(ATTRIBUTE,PTYPE,LTYPE), EMPQJ1(SEX) and EMPQJ2(SEX,TIME,STATE).

2. Scan the relation EMP1, and prepare a set of all the different values of the CA SEX. Obviously, this list in our example contains two objects: M and F. These values are the objects of the resulting TOR EMPQJ. Record them in the relation EMPQJ1.

3. The following steps are aimed to record the right values in the relation EMPQJ2. They are described for the object SEX=M of EMPQJ.

   a. Scan the relation EMP1 again, and prepare the list of all the objects in the original TOR EMP, having SEX=M.

   b. With this list, scan the relations EMP2 and EMP3 simultaneously, and locate the first time point at which some object in this list has a non NULL value at least in one of them. Such a situation indicates that the object SEX=M exists at this time point.

   c. If no such time point is found at all, it means that this object of the new TOR does not exist at all. Therefore, delete this object from EMPQJ1, and proceed to the next object of the new TOR (SEX=F).

d. If such a time point t is found, for some object in EMP, say OBJEMP, then record the tuple SEX=M, TIME=t and STATE=1 in EMPQJ2.

e. Continue to scan EMP2 and EMP3 simultaneously, looking for a possible NULL values in both of them for OBJEMP at some time point. If no such a case is found, then nothing more has to be recorded for SEX=M in EMPQJ2.

f. If NULL values were found in EMP2 and EMP3 for OBJEMP at some time point t_1, then look for another possible object of EMP having SEX=M, that exists at t_1. If no such an object is found, record the tuple SEX=M, TIME=t_1 and STATE=NULL in EMPQJ2. Now, keep scanning, looking for a possible object of EMP, having SEX=M, that exists at a later time point. If found, proceed as before. If not, proceed to the next object of EMPQJ (if any).

g. If NULL values were found in EMP2 and EMP3 for OBJEMP at some time point, but another object with SEX=M exists at this time point, then SEX=M still exists in EMPQJ. Denote the new object by OBJEMP, and return to step (e) above.

In this specific example, the new TOR contains two objects, M and F, and both of them exist in the TOR from the first time point included in the database (800101). Different cases are demonstrated in some of the remaining PROJECT operations discussed later in this section.

The procedure that corresponds to a PROJECT operation that projects only non-key CA(s) is a complicated one, since all the relations representing the original TOR's VAs are scanned simultaneously in order to identify the time points at which the new objects exist in the new TOR, and those at which they do not exist. In this case, the temporal differentiation of attributes and our way to indicate non existence of objects do not simplify the creation of the new TOR, and may well do the opposite.

QUERY 6.5 is an example of projecting all non-key CAs of a TOR by a PROJECT operation, omitting the key. It corresponds to case 11 in Table 6-1, in which all non-key CAs are projected onto the resulting TOR, and no VA is projected. As it is analogous to, though more complicated than, QUERY 6.4, it is briefly analyzed, without repeating the details of its evaluation.

QUERY 6.5

```
PROJECT EMP
ONTO NAME,SEX
INTO EMPQK
```

The objects of the new TOR created by this query are all the combinations of NAMEs and SEXs in the TOR EMP. Again, each time slice of the resulting TOR should be equivalent to the result of this PROJECT operation on the corresponding time slice of the operand in order to satisfy the strong correctness criterion. The relations representing the result of QUERY 6.5 are included in Table 6-4.

Let us now examine this operation with respect to the strong correctness criterion. Table 6-5 presents a snapshot from the resulting TOR EMPQK at 800501. Similarly, Table 6-6 presents a snapshot taken from the original TOR EMP at 800501.

Comparing Tables 6-5 and 6-6 leads to the observation that they actually contain the same information about the combinations of NAMEs and SEXs that exist in the TOR EMP at 800501. In examining the two tables, one should keep in mind the specific role of the VA STATE that was added to the resulting TOR.

EMPQK(ATTRIBUTE,PTYPE,LTYPE)

ATTRIBUTE	PTYPE	LTYPE
NAME	C20	1
SEX	C1	1
STATE	I2	3

EMPQK1(NAME,SEX)

NAME	SEX
MIKE	M
MARY	F
DAVID	M
HENRY	M
ALICE	F
OSCAR	M
SUSAN	F

EMPQK2(NAME,SEX,TIME,STATE)

NAME	SEX	TIME	STATE
MIKE	M	800101	1
MARY	F	810210	1
DAVID	M	800601	1
DAVID	M	820508	NULL
DAVID	M	830415	1
HENRY	M	800101	1
ALICE	F	810101	1
OSCAR	M	800101	1
SUSAN	F	800101	1

Table 6-4: The Relations Representing the Result of QUERY 6.5

The Key	The CA	The VA
NAME	SEX	STATE
MIKE	M	1
MARY	F	NULL
DAVID	M	NULL
HENRY	M	1
ALICE	F	NULL
OSCAR	M	1
SUSAN	F	1

Table 6-5: A Snapshot from the Result of QUERY 6.5 at 800501

The Key	The CAs		The VAs	
EMPNO	NAME	SEX	DEPTNO	JOBCLS
10010	MIKE	M	3	4
10005	MARY	F	NULL	NULL
10050	DAVID	M	NULL	NULL
10030	HENRY	M	2	3
10080	ALICE	F	NULL	NULL
10025	OSCAR	M	4	1
10090	SUSAN	F	4	4

Table 6-6: A Snapshot from the TOR EMP at 800501

The answer to QUERY 6.5 satisfies the strong correctness criterion. As explained in the remainder of this section, so do all other PROJECT operations that do not preserve the key, leading to the observation that all PROJECT operations satisfy the strong correctness criterion.

The remaining cases in Table 6-1 present implementational problems that are similar to those already discussed in this section. Therefore, we will present examples to these cases, without elaborating on their implementations.

Case 12 in Table 6-1 is demonstrated in this discussion by QUERY 6.6.

QUERY 6.6

```
PROJECT EMP
ONTO JOBCLS
INTO EMPQL
```

In QUERY 6.6, information about the various JOBCLSs in the organization is requested. The objects of the new TOR EMPQL will be the various values of JOBCLS included in the TOR EMP for its various objects (employees). For each of these values, the TOR EMPQL will indicate the periods of time in which it exists in the TOR. As the only attribute of this TOR, JOBCLS will be its key. The VA STATE will again be added to the resulting TOR, to allow the indication of the existence of each value of JOBCLS at any point of time.

The conceptual creation of the new TOR EMPQL is achieved by first removing the attribute JOBCLS from the TOR EMP, and then rearranging it by deleting tuples that became duplicates in the new TOR, even though they belonged to different objects in the original TOR. For example, employee 10030 started his work in the organization at 800101 with JOBCLS 3. Then, at 820101 he was promoted to JOBCLS 2. However, at that time, there were other employees with JOBCLS 3, and, in fact, for any point of time starting at 800101, at least one employee in the organization had JOBCLS 3. Therefore,

the object JOBCLS=3 in the TOR EMPQL exists at all time points since 800101. As opposed to JOBCLS 3, employees with JOBCLS=4 do not exist in the TOR EMP after 811014, and therefore the object JOBCLS=4 ceases to exist in the TOR EMPQL at 811015.

The relations representing the new TOR EMPQL, answering QUERY 6.6, are included in Table 6-7. Table 6-8 presents a snapshot taken from the resulting TOR EMPQL at the arbitrary day 811230. This table should be compared to the result of the equivalent regular PROJECT on the time slice of the TOR EMP at 811230. This comparison will demonstrate that QUERY 6.6 satisfies the strong correctness criterion.

QUERY 6.7 corresponds to case 13 in Table 6-1. The key is dropped in this query, and a combination of one CA and one VA is projected to the new TOR.

QUERY 6.7

```
PROJECT EMP
ONTO SEX DEPTNO
INTO EMPQM
```

The objects of the new TOR EMPQM are all the combinations of DEPTNOs and SEXs that exist at some time point in the TOR EMP. Each combination like this will be included in the resulting TOR, with time stamps associated to its VA STATE, which indicate precisely when it exists and when it does not.

As usual, the purpose of this query can be better understood by examining

EMPQL(ATTRIBUTE,PTYPE,LTYPE)

ATTRIBUTE	PTYPE	LTYPE
JOBCLS	I2	1
STATE	I2	3

EMPQL1(JOBCLS)

JOBCLS
4
3
2
1

EMPQL2(JOBCLS,TIME,STATE)

JOBCLS	TIME	STATE
4	800101	1
4	811015	NULL
3	800101	1
2	810101	1
1	800101	1

Table 6-7: The Relations Representing the Result of QUERY 6.6

JOBCLS	STATE
4	NULL
3	1
2	1
1	1

Table 6-8: A Snapshot from the Result of QUERY 6.6 at 811230

a random snapshot from it. Table 6-9 presents a possible snapshot of the TOR EMPQM (say, at 811010).

SEX	DEPTNO	STATE
M	1	1
M	2	1
M	3	NULL
M	4	1
F	2	1
F	3	1
F	4	1

Table 6-9: A Snapshot from the Result of QUERY 6.7 at 811010

Table 6-9 indicates which combinations of SEXs and DEPTNOs exist in the TOR EMP at 811010. The NULL value in the VA STATE for the object SEX=M and DEPTNO=3 indicates that there is no male employee in department 3 at 811010. The new TOR EMPQM conceptually consists of such snapshots at all its time points. Like the previous examples, the current one also satisfies the the strong correctness criterion.

The regular relations representing the TOR EMPQM are included in Table 6-10. Consider, for instance, the object SEX=M and DEPTNO=3 in this table. At 800101, the STATE of this object is 1 (namely, it exists), since the male employee 10010 is working in department 3. At 810215, employee 10010 leaves department 3, and at that time point there is no other male in this department. Therefore, the STATE of this object becomes NULL, until 820701, the day at which employee 10030 (also, a male) joins the department. This employee leaves department 3 at 830508, and from this day there is no male in the department, which implies the value NULL for the VA STATE of this object, at the period of time starting at 830508.

QUERY 6.8 illustrates case 14 in Table 6-1. In this case, all the VAs are projected onto the resulting TOR, but none of the CAs. The objects of the new TOR EMPQN are all the combinations of DEPTNOs and JOBCLSs that exist at any time point in the TOR EMP.

QUERY 6.8

```
PROJECT EMP
ONTO DEPTNO,JOBCLS
INTO EMPQN
```

The relations representing the TOR EMPQN are included in Table 6-11. Consider, for instance, the object DEPTNO=2 and JOBCLS=2 as recorded in the new relation EMPQN2. This object exists for the first time at 820101, the day at which employee 10030 is promoted to JOBCLS=2, while working for department 2. Then, at 820701 this employee leaves department 2. As there is no other

EMPQM(ATTRIBUTE,PTYPE,LTYPE)

ATTRIBUTE	PTYPE	LTYPE
SEX	C1	1
DEPTNO	I2	1
STATE	I2	3

EMPQM1(SEX,DEPTNO)

SEX	DEPTNO
M	1
M	2
M	3
M	4
F	2
F	3
F	4

EMPQM2(SEX,DEPTNO,TIME,STATE)

SEX	DEPTNO	TIME	STATE
M	1	800601	1
M	1	820508	NULL
M	1	830415	1
M	2	800101	1
M	3	800101	1
M	3	810215	NULL
M	3	820701	1
M	3	830508	NULL
M	4	800101	1
F	2	810210	1
F	3	810101	1
F	4	800101	1

Table 6-10: The Relations Representing the Result of QUERY 6.7

employee in this department with JOBCLS=2 on this day, the object DEPTNO=2 and JOBCLS=2 ceases to exist on this day. Then, at 820115, another employee in department 2, employee 10010, is promoted to JOBCLS=2, and therefore this object exists again on this day. This employee never leaves the department, neither is he further promoted. Therefore the object DEPTNO=2 and JOBCLS=2 exists until now.

EMPQN(ATTRIBUTE,PTYPE,LTYPE)

ATTRIBUTE	PTYPE	LTYPE
DEPTNO	I2	1
JOBCLS	I2	1
STATE	I2	3

EMPQN1(DEPTNO,JOBCLS)

DEPTNO	JOBCLS
3	4
3	3
2	3
2	2
1	3
1	2
3	2
3	1
2	1
4	1
4	4
4	3

EMPQN2(DEPTNO,JOBCLS,TIME,STATE)

DEPTNO	JOBCLS	TIME	STATE
3	4	800101	1
3	4	810201	NULL
3	3	810201	1
3	3	810215	NULL
2	3	800101	1
2	2	820101	1
2	2	820701	NULL
2	2	821015	1
1	3	800601	1
1	3	820508	NULL
1	2	830415	1
3	2	810101	1
3	1	830304	1
3	1	830508	NULL
2	1	830508	1
4	1	800101	1
4	4	800101	1
4	4	811015	NULL
4	3	811015	1

Table 6-11: The Relations Representing the Result of QUERY 6.8

In a more general case, more than two VAs can be projected onto the new TOR. This PROJECT operation, like all other operations included in this section, satisfies the strong correctness criterion.

Case 15 in Table 6-1 is illustrated by QUERY 6.9. The objects of the resulting TOR EMPQO are all combinations of SEX, DEPTNO and JOBCLS, included at some time point in the original TOR EMP. Following the convention, the new TOR's key contains all its attributes.

QUERY 6.9

```
PROJECT EMP
ONTO SEX,DEPTNO,JOBCLS
INTO EMPQO
```

The relations representing the TOR EMPQO are included in Table 6-12. Consider the history of the object SEX=M, DEPTNO=2 and JOBCLS=2 in this table. As shown in the relations EMP1, EMP2 and EMP3, the first day in which this object exists is 820101, at which employee 10030, a male working in department 2, has been promoted to JOBCLS 2. This object ceases to exist at 820701, since this employee left department 2 at that day. The object does not exist in the new TOR until 821015, the day in which employee 10010, a male working in department 2 on that day, has been promoted to JOBCLS 2. From that day, there is no change in both the department and the JOBCLS of this employee, and therefore no further information has to be recorded for the object SEX=M, DEPTNO=2 and JOBCLS=2 in the new TOR EMPQO.

EMPQO(ATTRIBUTE,PTYPE,LTYPE)

ATTRIBUTE	PTYPE	LTYPE
SEX	C1	1
DEPTNO	I2	1
JOBCLS	I2	1
STATE	I2	3

EMPQQ1(SEX,DEPTNO,JOBCLS)

SEX	DEPTNO	JOBCLS
M	3	4
M	3	3
M	2	3
F	2	3
M	2	2
M	1	3
M	1	2
M	3	2
F	3	2
M	3	1
M	2	1
M	4	1
M	4	4
M	4	3

EMPQQ2(SEX,DEPTNO,JOBCLS,TIME,STATE)

SEX	DEPTNO	JOBCLS	TIME	STATE
M	3	4	800101	1
M	3	4	810201	NULL
M	3	3	810201	1
M	3	3	810215	NULL
M	2	3	800101	1
M	2	3	821015	NULL
F	2	3	810210	1
M	2	2	820101	1
M	2	2	820701	NULL
M	2	2	821015	1
M	1	3	800601	1
M	1	3	820508	NULL
M	1	2	830415	1
M	3	2	820701	1
M	3	2	830304	NULL
F	3	2	810101	1
M	3	1	830304	1
M	3	1	830508	NULL
M	2	1	830508	1
M	4	1	800101	1
F	4	4	800101	1
F	4	4	811015	NULL
F	4	3	811015	1

Table 6-12: The Relations Representing the Result of QUERY 6.9

The PROJECT operation illustrated by QUERY 6.9, like all other PROJECT operations, satisfies the strong correctness criterion.

Case 16 in Table 6-1 can be demonstrated by QUERY 6.10. The objects of the new TOR EMPQP are all combinations of NAMEs, SEXs and DEPTNOs included at any time point in the original TOR EMP.

QUERY 6.10

```
PROJECT EMP
ONTO NAME,SEX,DEPTNO
INTO EMPQP
```

Case 17 in Table 6-1 describes a PROJECT operation in which all the non-key attributes are projected onto the new TOR. QUERY 6.11 belongs to this category. The objects of the new TOR EMPQQ are all the combinations of NAMEs, SEXs, DEPTNOs and JOBCLSs that exist at any time point in the original TOR EMP.

QUERY 6.11

```
PROJECT EMP
ONTO NAME,SEX,DEPTNO,JOBCLS
INTO EMPQQ
```

Case 18 in Table 6-1 is an "empty" case that does not require any discussion.

Cases 19-27 in Table 6-1 repeat cases 10-18, with the difference that only part of the key attributes is omitted. This difference, however, does not change the nature of these PROJECT operations. Any key attribute that is projected in such a PROJECT operation is simply treated as a CA, and loses its uniqueness as a prime attribute, since the key is not preserved in its entirety.

6.3. Summary of the PROJECT Operation

In this chapter, 27 different cases of TOR projection were identified (Table 6-1). Two of these cases (case 1 and case 18) are trivial cases, while 25 of them actually define meaningful PROJECT operations. After analyzing these cases, we can determine the distinction between PROJECT operations that preserve the original key, and those that do not, as capturing the fundamental classes of PROJECT operations.

All the cases in which the key is preserved (cases 1 - 9) are basically the same. They maintain the original objects' identities, and just drop some attribute(s) from each of them. Not only the conceptual semantics of all PROJECT operations preserving the key are the same, but their implementations are very similar too.

The PROJECT operations that preserve the key do not cause any problem in their conceptual definition and in their design. This is not true for those operations that do not preserve the key. They create new objects, consisting of all the projected attributes, and giving new meaning to the resulting TOR, compared to the original's. The conceptual semantics of such a PROJECT operation is still straightforward. This PROJECT, executed on every time slice of the operand, produces new valid relations (after deleting the duplicate tuples that were created as a result of dropping the key). The new key of these relations consists of all their attributes. The new TOR consists of all these relations as its various time slices.

The problem with these operations starts when combining all these

separate time slices into the new TOR (cube). Usually, they do not contain exactly the same objects. In the cube, however, an object that exists at one time point, automatically has an entire horizontal slice containing its tuples at all time points. We proposed to add an artificial VA, called STATE, to the resulting TOR of such an operation, to allow the indication of possible non-existence of objects at some time points. This may not be the optimal solution, and more research is needed to identify other possible, and may be better, solutions.

This concludes the discussion covering the PROJECT operation. Considering the effect of the temporal differentiation of attributes on these operations, it is clear that it makes the implementation of the operations preserving the key very simple, by allowing us to ignore the relations representing attributes that are not projected. On the other hand, in all the queries that deal with PROJECT operations that do not preserve the key, none of the relations representing the existing TOR can simply be copied to the relations representing the new TOR, since the new TOR contains new kind of objects. However, the temporal differentiation of attributes is sometimes beneficial even in these cases, by allowing us to concentrate on the attributes that are included in the new TOR, and ignore the others. This, by itself, is an advantage in the design of any temporal operation. Unfortunately, in some cases this concept is not beneficial at all (e.g., when only non-key CAs are projected).

The syntax of the PROJECT operation, supported by our TDMS, is consistent with the restriction stated before, according to which a query can slice the

cube only in one dimension at a time. Therefore, a mixture of PROJECT and SELECT operations should be achieved through a sequence of queries that will produce the same result as the one obtained by one equivalent query. It should be mentioned again, that this restriction does not impose any limitation on the capabilities of the TDMS, but is needed to make the design clearer and simpler. In creating a sequence of queries aimed at deriving an answer to some question, the user should also be aware of the possible non-commutativities in executing some queries, and execute the temporal relational algebra operations in the order that fits the desired results.

6.4. The Time Projection Operation

A PROJECT operation that does not preserve the entire key may be interpreted in more than one way. Earlier in this chapter, we have chosen the interpretation according to which every time slice of the resulting TOR is equivalent to the same PROJECT on the corresponding time slice of the operand. This interpretation conforms to the strict "naturalness" criteria, and reflects the meaning of such a PROJECT operation in the regular relational model. It is also consistent with the general framework of the temporal relational algebra operations, developed in Chapter 4. In this section, however, we develop another type of operation, the Time Projection, as an alternative to the interpretation adopted earlier for such operations.

The Time Projection does not analyze the data stored in the TOR through its time slices, but considers the whole temporal pattern of the resultant objects. Two objects in the original TOR are "merged" to the same object in the resulting TOR, only if their projected attributes exhibit identical

temporal pattern. Otherwise, they are considered different objects in the new TOR as well.

The definition of the time projection operation at the external level is as follows:

Given a TOR T, where T is the union of n flat relations:

$$T = \text{UNION}_{i=1}^{n}(R_i),$$

then, the new TOR S, containing the result of a time projection operation, with respect to a set of attributes SA, is:

$$S = \text{UNION}_{i=1}^{n}(S_i)$$

where each S_i contains the data of those objects which represent all different temporal patterns with respect to the set of attributes SA.

The definition of this operation in terms of the internal view of the operand and the resulting TOR, is as follows:

$$\text{TIME PROJECTION}[V] = \text{UNION}_{i=1}^{k}(OB_i)$$

where OB_i is a qualifying object, namely an object that contains a unique pattern of the combination of the projected attributes (only these attributes are included in the new objects).

In implementing the Time Projection, one faces the problem of creating possible duplicate tuples in various time slices. These tuples belong to different objects of the resultant TOR, and their inclusion in it is caused by the nature of this operation, which, as opposed to the "natural" temporal PROJECT, does not conceptually operate on the basis of the time-slices of the operand. To solve this problem, a new attribute is added to the resulting

TOR, and this new attribute is the new TOR's key, by which the objects are identified. This attribute is needed for the unique identification of objects in the new TOR. An obvious choice is to identify the objects by serial numbers, i.e, the first object is No 1, the second is No 2, etc. We label the new attribute IDENT.

The time projection satisfies only the weak correctness criterion, as this operation does not conceptually operate on one time slice at a time, but on the entire cubic view of the TOR as such. Therefore, when applied to a TOR that happens to be a single time slice, it produces the same result as the regular PROJECT operation on the snapshot representation of this flat TOR. However, when applied to a TOR that contains data for more than one time point, it produces a new TOR whose various time slices are not equivalent to the results of this PROJECT operation on the corresponding time slices of the operand. A snapshot taken from the resulting TOR possibly contains tuples that are duplicates in their content, and only the new attribute IDENT, added to them, makes them, different. However, when applying this PROJECT to the snapshot from the operand, the duplicate tuples are actually deleted, producing a relation that is different from the one produced by a snapshot from the resulting TOR.

The Time Projection operation is briefly discussed in this section, using some of the queries presented earlier in section 6.2, applied, for demonstration purposes, to the TOR NEWEMP. This TOR has the same definition as EMP, but different data, allowing the presentation of the different variations of the time projection. The internal view of the TOR NEWEMP is presented in Table 6-13.

NEWEMP(EMPNO,NAME,SEX,DEPTNO,JOBCLS)

EMPNO	NAME	SEX	DEPTNO	JOBCLS
10010	MIKE	M	800101 3	800101 4
			810215 2	810201 3
				821015 2
10005	MARY	F	800101 3	800101 4
			810215 2	810201 3
				821015 2
10050	MINE	M	800101 3	800101 3
			810215 2	820101 2
				830304 1
10030	HENRY	M	800101 2	800101 3
			820701 3	820101 2
			830508 2	830304 1
10080	SUSAN	F	810101 3	810101 2
10025	OSCAR	M	800101 4	800101 1
10090	SUSAN	F	800101 4	800101 4
				811015 3

Table 6-13: The Internal View of the TOR NEWEMP

Let us start to analyze the time projection, with QUERY 6.12, which is the time projection version of QUERY 6.6.

QUERY 6.12

```
CREATE TIME PROJECTION
FROM NEWEMP
ONTO JOBCLS
INTO TPRC
```

In QUERY 6.12, information about the various JOBCLSs in the organization is requested. The objects of the new TOR TPRC are determined by the various different patterns of JOBCLS included in the TOR NEWEMP for its various objects (employees). The relations representing the new TOR TPRC answering QUERY 6.12 are included in Table 6-14.

TPRC(ATTRIBUTE,PTYPE,LTYPE)

ATTRIBUTE	PTYPE	LTYPE
IDENT	I2	1
JOBCLS	I2	3

TPRC1(IDENT,JOBCLS)

IDENT
1
2
3
4
5

TPRC2(IDENT,TIME,JOBCLS)

IDENT	TIME	JOBCLS
1	800101	4
1	810201	3
1	821015	2
2	800101	3
2	820101	2
2	830304	1
3	810101	2
4	800101	1
5	800101	4
5	811015	3

Table 6-14: The Relations Representing the Result of QUERY 6.12

The new TOR TPRC has five objects (compared to seven in the original TOR NEWEMP), since the original objects (employees) 10010 and 10005, as well as 10050 and 10030, have identical JOBCLS patterns.

The procedure to create the new TOR TPRC, answering QUERY 6.12, is as follows:

1. Create the descriptive relation TPRC(IDENT,JOBCLS). The new attribute IDENT is the key. In this case, the new TOR has no CAs, and JOBCLS is its only VA.

2. Scan the relation NEWEMP3, describing the VA JOBCLS in the TOR NEWEMP, and determine how many different patterns of JOBCLS values are included in it. As mentioned earlier, the original TOR NEWEMP has five different patterns of JOBCLS. Therefore, the new TOR has five objects: 1,2,...,5. Record them in the relation TPRC1(IDENT), describing the CAs in the new TOR TPRC.

3. The information of the five different patterns should be copied from NEWEMP3 to TPRC2(IDENT,TIME,JOBCLS), associating each tuple being copied with the correct value of IDENT. Whenever two or more objects have the same pattern in NEWEMP3, only one of them is copied to TPRC2.

QUERY 6.13 presents the time projection version of QUERY 6.7. The key is dropped in this query, and a combination of one CA and one VA is projected onto the new TOR.

QUERY 6.13

```
CREATE TIME PROJECTION
FROM NEWEMP
ONTO SEX DEPTNO
INTO TPRD
```

The objects of the new TOR TPRD are determined by all the combinations of

different patterns of DEPTNOs and SEXs that exist in the TOR NEWEMP. Each different combination like this forms an object in the resulting TOR.

The regular relations representing the TOR TPRD are included in Table 6-15. Two objects of NEWEMP are "merged" to one in TPRD: employee 10010 and employee 10050, since both of them are male, and have the same pattern in the variation of their DEPTNOs. Note also that employee 10005 is not merged with them despite her identical DEPTNO pattern, since she has a different SEX. The same is true for employees 10025 compared to 10090.

QUERY 6.13 is resolved by the following procedure:

1. Create the new descriptive relation TPRD. IDENT is, as usual, its key. SEX is a CA in this TOR and DEPTNO is a VA in it.

2. The objects of the new TOR are determined by all combinations of different patterns of SEX and DEPTNO in the original TOR NEWEMP. Therefore, scan NEWEMP1 (representing SEX) and NEWEMP2 (representing DEPTNO), and prepare a list of the objects of this TOR, representing all these different combinations. Allocate the IDENT values to these objects in the new TOR TPRD, and record them in TPRD1, together with their SEXs.

3. Copy from NEWEMP2 to TPRD2 all the information of the objects included in the list, and associate the correct value of IDENT with each tuple being copied.

4. Ignore the relation NEWEMP3.

QUERY 6.14 is the time projection version of QUERY 6.8. All the VAs are projected onto the resulting TOR, and none of the CAs.

TPRD(ATTRIBUTE,PTYPE,LTYPE)

ATTRIBUTE	PTYPE	LTYPE
IDENT	I2	1
SEX	C1	2
DEPTNO	I2	3

TPRD1(IDENT,SEX)

IDENT	SEX
1	M
2	F
3	M
4	F
5	M
6	F

TPRD2(IDENT,TIME,DEPTNO)

IDENT	TIME	DEPTNO
1	800101	3
1	810215	2
2	800101	3
2	810215	2
3	800101	2
3	820701	3
3	830508	2
4	810101	3
5	800101	4
6	800101	4

Table 6-15: The Relations Representing the Result of QUERY 6.13

QUERY 6.14

```
CREATE TIME PROJECTION
FROM NEWEMP
ONTO DEPTNO,JOBCLS
INTO TPRE
```

The objects of the new TOR TPRE are determined by all the combinations of different patterns of the VAs DEPTNO and JOBCLS in the TOR NEWEMP. The relations representing the TOR TPRE are included in Table 6-16.

TPRE(ATTRIBUTE,PTYPE,LTYPE)

ATTRIBUTE	PTYPE	LTYPE
IDENT	I2	1
DEPTNO	I2	3
JOBCLS	I2	3

TPRE1(IDENT)

IDENT
1
2
3
4
5
6

TPRE2(IDENT,TIME,DEPTNO)

IDENT	TIME	DEPTNO
1	800101	3
1	810215	2
2	800101	3
2	810215	2
3	800101	2
3	820701	3
3	830508	2
4	810101	3
5	800101	4
6	800101	4

TPRE3(IDENT,TIME,JOBCLS)

IDENT	TIME	JOBCLS
1	800101	4
1	810201	3
1	821015	2
2	800101	3
2	820101	2
2	830304	1
3	800101	3
3	820101	2
3	830304	1
4	810101	2
5	800101	1
6	800101	4
6	811015	3

Table 6-16: The Relations Representing the Result of QUERY 6.14

Note that employees 10010 and 10005 are projected into one object for this query. Also note that employees 10050 and 10030 have the same JOBCLS pattern, but since they have different DEPTNO patterns they are projected into two objects in the new TOR.

Consider QUERY 6.15, which is the time projection version of QUERY 6.9.

QUERY 6.15

```
CREATE TIME PROJECTION
FROM NEWEMP
ONTO SEX,DEPTNO,JOBCLS
INTO TPRF
```

The objects of the resulting TOR TPRF are determined by all combinations of the different patterns of SEX, DEPTNO, and JOBCLS in the TOR NEWEMP. Checking the information in this TOR, leads to the conclusion that no two objects in it have the same combination in their patterns of these attributes. Therefore, all the operand's objects will be different objects in the new TOR too, and in practice all the information included in NEWEMP2 and NEWEMP3 will be copied to the new TOR, with no change, except the addition of the identification numbers.

QUERY 6.16 is the time projection version of QUERY 6.11. In this query, all the non-key attributes are projected.

QUERY 6.16

```
CREATE TIME PROJECTION
FROM NEWEMP
ONTO NAME,SEX,DEPTNO,JOBCLS
INTO TPRH
```

Considering the content of NEWEMP, one can easily verify that all its objects maintain their uniqueness, as well as their data, in the new TOR.

The time projection operation provides an alternative to the meaning of executing a PROJECT operation that does not preserve the entire key. The user may choose the operation that fits his needs, either this operation or the temporal PROJECT.

Chapter 7

The Temporal JOIN Operation

The JOIN operation is used to combine information from two relations, and merge them into a new relation. Generally, two participating relations have some common attribute(s), either explicitly by names, or implicitly by content. These attributes serve as the tool to combine the corresponding tuples from the two relations.

The temporal JOIN operation has not yet been fully and satisfactorily defined [Ariav 83a], [Clifford 85a], [Snodgrass 85] and [Gadia 84]. However, within our framework of implementing the temporal relational algebra operations (see Chapter 4), its conceptual definition in our model is fairly straightforward: a TOR J is a result of a JOIN operation with two existing TORs A and B, if all the possible snapshots of J are the results of regular JOIN operations with the corresponding snapshots of A and B.

The definition of the temporal JOIN at the external level is as follows:

Given two TORs K and L, where:

$$K = \text{UNION}_{i=1}^{n}(R_i),$$

$$L = \text{UNION}_{i=1}^{n}(S_i)$$

where R_i and S_i are the slices of time point i.

then, the new TOR M, containing the result of the temporal JOIN of K and L is:

$$M = \text{UNION}_{i=1}^{n}(T_i)$$

where each T_i is the result of the regular JOIN:

$$T_i = \text{JOIN}(R_i, S_i)$$

The difficulties involved in implementing the temporal JOIN are substantial, since two cubes are merged here, rather than two flat tables. The implementation of this operation in the TDMS critically depends upon the types of the common attributes in the joined TORs. It takes full advantage of the temporal differentiation of attributes, by decomposing the operation into a sequence of manipulations of corresponding pairs of relations, representing the operands. Typically, most of these manipulations are straightforward, with the exception of the manipulation of the common attribute. Therefore, this decomposition isolates the difficulties involved in this operation, and localize them.

The analysis of the JOIN operation in this chapter is limited to those cases in which there is only one common attribute to the two operands. For such operations, a comprehensive framework of classifying the various cases is provided by Table 7-1.

Each "junction" in Table 7-1 represents a different case with respect to the implementation of the JOIN. Since the temporal JOIN is defined as a commutative operation, there are only six basic cases, as follows:

1. The common attribute is the key in both operands (key - key JOIN).
2. The common attribute is the key in one TOR, and a CA in the other (key - CA JOIN).

The types of the common attribute in the operands	KEY	CA	VA
KEY	7.1	7.2	7.3
CA	7.2	7.4	7.5
VA	7.3	7.5	7.6

Table 7-1: The Categories of the JOIN Operation

3. The common attribute is the key of one TOR, and a VA in the other (key - VA JOIN).

4. The common attribute is a CA in both operands (CA - CA JOIN).

5. The common attribute is a CA in one TOR, and a VA in the other (CA - VA JOIN).

6. The common attribute is a VA in both operands (VA - VA JOIN).

Each of the six basic cases is discussed in a separate section. The section number is written in the "box", corresponding to this particular case. All the cases included in Table 7-1, are analyzed through comprehensive examples. The semantics of each case is explained, followed by the description of the implementation. Then, the correctness of the operation is demonstrated, using the regular JOIN with two arbitrary snapshots of the operands, and comparing them to the corresponding snapshot taken from the resulting TOR. Note that in the following discussion we use the terms "first"

and "second" to identify conveniently the operands of the temporal JOIN. It should by no means be interpreted as if this operation is not a commutative one.

The demonstration of the correctness of the JOIN operation requires a correct interpretation of NULL values. A snapshot taken from either operand may have objects that contain NULL values for all their VAs, indicating their inexistence, which should be inherited by the result of the JOIN. However, the JOIN operation does not "know" the meaning of NULL values assigned to all the VAs of an object, and it matches such tuples with tuples of the other snapshot, possibly containing non-NULL values for their VAs. Therefore, in the resulting relation non-existing objects do not contain NULL values for all their VAs, but only for those VAs which are originally NULL in their snapshot. In order to correct this situation, an adjustment is made in which before executing the JOIN, all the tuples that contain NULL values for all their VAs in either snapshot (indicating that they practically do not exist) should be deleted, and then the JOIN should be executed. This adjustment does not change these snapshots, since only non-existing objects are deleted. One can even find a deeper reason for this problem. The only reason for the introduction of NULL values is the nature of the cube, that requires some means to indicate non existence values at a specific time point. Once a snapshot is taken, the NULL values are not needed any more, since at a specific time point, either the value exists or it does not. Therefore, the real task of NULL values assigned to all the VAs of some object at some time point is to imply the deletion of this object from a snapshot taken at this time point, which underlies the "adjustment" suggested above.

In applying this adjusted procedure to any result of a temporal JOIN operation, it turns out that the snapshot taken from the resulting TOR is always equivalent to the result of joining the corresponding snapshots of the operands (according to the definition included in Chapter 4). Therefore, the temporal JOIN is an operation that can be defined to satisfy the strong correctness criterion.

7.1. key - key JOIN

Let us start with perhaps the simplest JOIN operation, one that merges two TORs having exactly the same key, and no common attributes outside the key.

QUERY 7.1

```
JOIN EMP WITH SAL
INTO EMPSAL
```

The TORs EMP and SAL are included in Appendix A. They have the same key EMPNO, which is also their only common attribute. Therefore, merging them into one TOR is fairly simple.

The semantics of this query is simple too. It creates a new TOR, containing for each employee all his attributes from both EMP and SAL. Tables 7-2 and 7-3 present two random snapshots of the operands (at 830101). Note, that according to the adjustments, discussed above, tuples containing NULL values in all their VAs have been deleted from these tables.

The Key	The CAs		The VAs	
EMPNO	NAME	SEX	DEPTNO	JOBCLS
10010	MIKE	M	2	2
10005	MARY	F	2	3
10030	HENRY	M	3	2
10080	ALICE	F	3	2
10025	OSCAR	M	4	1
10090	SUSAN	F	4	3

Table 7-2: A Snapshot from the TOR EMP at 830101

The Key	The VA
EMPNO	SALARY
10010	22100
10005	22500
10030	23500
10080	24000
10025	32300
10090	21200

Table 7-3: A Snapshot from the TOR SAL at 830101

The JOIN operation, included in QUERY 7.1, conceptually joins each pair of such corresponding snapshots from the two operands, to form new snapshots, containing their combined information. Then, all these snapshots are combined together, to form the cubic view of this new TOR. This is the conceptual structure of this JOIN. The result of joining the two snapshots, presented in Tables 7-2 and 7-3, is included in Table 7-4.

Key	The CAs		The VAs		
EMPNO	NAME	SEX	DEPTNO	JOBCLS	SALARY
10005	MARY	F	2	3	22500
10010	MIKE	M	2	2	22100
10025	OSCAR	M	4	1	32300
10030	HENRY	M	3	2	23500
10080	ALICE	F	3	2	24000
10090	SUSAN	F	4	3	21200

Table 7-4: The Result of Joining the Snapshots of EMP and SAL at 830101

Table 7-4 presents the salary of every employee in the same relation with his data, as included in EMP. The new TOR consists of such snapshots for all time points.

In order to describe the general procedure to evaluate such a JOIN operation, assume that two TORs, RELA and RELB, are joined to form a new TOR RELJA. These two TORs have the same key, denoted by (key_attributes). Moreover, the key attributes are the two TORs' only common attributes. The TOR RELA has n VAs, and is represented by the relations RELA, $RELA_1$, ..., REL_{n+1}. The TOR RELB has m VAs, and is represented by the relations RELB, $RELB_1$, ..., REL_{m+1}. The procedure to create the new TOR RELJA is included in the following Algorithm 7.1:

1. The descriptive relation RELJA is the result of the following UNION operation:

   RELJA = UNION(RELA,RELB)

2. The relation $RELJA_1$ is The result of the following regular JOIN operation:

JOIN $RELA_1$ WITH $RELB_1$
INTO $RELJA_1$
WHERE $RELA_1$.(key_attributes)=$RELB_1$.(Key_attributes)

3. If VAR, one of the VAs in RELA, is represented in RELA by the relation $RELA_i$, then its values are copied to the relation $RELJA_i$ by the following JOIN operation:

JOIN $RELA_i$ WITH $RELJA_1$
INTO $RELJA_i$
USING $RELA_i$.(key_attributes),$RELA_i$.TIME,$RELA_i$.VAR
WHERE $RELA_i$.(key_attributes)=$RELJA_1$.(key_attributes)

4. The tuples of the relations $RELB_2$, ..., $RELB_{m+1}$ should be copied to the relations $RELJA_{n+2}$, ..., $RELJA_{n+m+1}$, through the same procedure as in step 3 above.

5. Each object O_{new} of the new TOR is a combination of two objects O_1 and O_2 of the two operands. NULL values should be recorded in all the VAs of O_{new} at all time points at which at least one of the original objects does not exist.

Note that in this particular case the algorithm covers any number of common attributes, as long as they form the key in the two operands.

The same procedure, applied to the TOR EMPSAL, created by joining the TORs EMP and SAL, is the following:

1. The descriptive relation EMPSAL is the result of the following UNION operation:

EMPSAL = UNION(EMP,SAL)

2. EMPSAL1 is the result of the following JOIN:

JOIN EMP1 WITH SAL1
INTO EMPSAL1
WHERE EMP1.EMPNO=SAL1.EMPNO

3. EMPSAL2 is the result of the following JOIN:

JOIN EMP2 WITH EMPSAL1
INTO EMPSAL2

```
USING EMP2.EMPNO,EMP2.TIME,EMP2.DEPTNO
WHERE EMP2.EMPNO=EMPSAL1.EMPNO
```

EMPSAL3 is the result of a similar JOIN, where EMP3 replaces EMP2.

4. The tuples from SAL2 (describing SALARY in SAL) should be copied to EMPSAL4 through the same procedure as described in step 3 above.

5. NULL values should be inserted in the VAs of non-existing objects, as inherited from the operands (see Algorithm 7.1).

In a simple JOIN as the one included in QUERY 7.1 (key - key JOIN), all the attributes maintain their original types in the new TOR. Therefore, the algorithm that creates the new representing relations does not have to deal with the problem of determining their types in the new TOR. This problem appears, however, in more complicated JOIN operations.

There is no need to present the regular relations representing the new TOR EMPSAL, as no new regular relation has been created (except the descriptive relation), and the existing regular relations that have been recombined to represent this new TOR are included in Appendix A. Note that in our database, the TORs EMP and SAL contain the same objects, and consequently these objects are also the objects of the new TOR EMPSAL. However, if this is not the situation, then only those objects that are included in both EMP and SAL would be included in EMPSAL.

This query is a good example demonstrating some of the advantages of the temporal differentiation of attributes. The creation of the new regular relations, representing the new TOR, has been achieved through a sequence of operations on the various relations representing the operands, thus

illustrating how the problem of implementing the temporal JOIN has been decomposed to a sequence of simple and manageable activities.

The new TOR EMPSAL, which is the result of joining EMP and SAL has the same CAs as EMP, and therefore its CAs are represented by the relation EMP1, describing the CAs of the TOR EMP. Its VAs are represented by EMP2, EMP3 and SAL2. Using these relations (see in Appendix A), we can now take a snapshot from the new TOR EMPSAL at 830101. This snapshot will be equivalent to the result of joining the corresponding snapshots from EMP and SAL, as represented in Table 7-4. The only difference will be a tuple of employee 10050, containing NULL value for all his VAs (and therefore neglectable) in one snapshot. This demonstrates that the definition of Key - Key JOIN conforms to the strong correctness criterion.

7.2. key - CA JOIN

QUERY 7.2 illustrates a JOIN operation in which the common attribute is the key in one TOR and a CA in the other.

QUERY 7.2

```
JOIN DRESS WITH EMP
INTO DRSEMP
```

The TOR EMP contains the data of each employee. The TOR DRESS describes the dressing rooms used by the employees at all time points. Different dressing rooms are allocated to men and women. Therefore, the attribute SEX is the key of this TOR. On the other hand, SEX is one of the CAs in the TOR EMP.

The purpose of QUERY 7.2 is to create a new TOR in which information about the dressing rooms used by the various employees, according to their SEXs, will be added to the data of these employees in the TOR EMP. The meaning of this query is illustrated by joining two arbitrary snapshots of the two operands. Tables 7-5 and 7-6 present the snapshots of the TORs EMP and DRESS at the arbitrary day 830101. Table 7-7 presents the result of the regular JOIN with the snapshots included in Tables 7-5 and 7-6.

Key	The CAs		The VAs	
EMPNO	NAME	SEX	DEPTNO	JOBCLS
10005	MARY	F	2	3
10010	MIKE	M	2	2
10025	OSCAR	M	4	1
10030	HENRY	M	3	2
10080	ALICE	F	3	2
10090	SUSAN	F	4	3

Table 7-5: A Snapshot from the TOR EMP at 830101

Key	VA
SEX	ROOM
M	M404
F	M610

Table 7-6: A Snapshot from the TOR DRESS at 830101

In Table 7-7, the attribute ROOM is added to the other attributes of each

Key	The CAs		The VAs		
EMPNO	NAME	SEX	DEPTNO	JOBCLS	ROOM
10005	MARY	F	2	3	M610
10010	MIKE	M	2	2	M404
10025	OSCAR	M	4	1	M404
10030	HENRY	M	3	2	M404
10080	ALICE	F	3	2	M610
10090	SUSAN	F	4	3	M610

Table 7-7: The Result of Joining Snapshots from EMP and DRESS at 830101

employee, as included in EMP, thus providing the information about his/her dressing room at 830101. The new TOR DRSEMP conceptually consists of such snapshots at all time points.

The common attribute, SEX, is the key of the TOR DRESS, and a CA in the TOR EMP. The type of such an attribute in the new TOR is determined by its type in the TOR in which it is not the key, and therefore it will be a CA in DRSEMP. The attribute ROOM which is a VA in DRESS will maintain its type in DRSEMP. In addition, if the TOR DRESS would have had some CAs, they would have been also CAs in DRSEMP. This situation is not caused by the temporal nature of the operands. It can be explained by the dependency theory in the regular relational model. There, it does not affect the implementation; here it does.

The creation of the new TOR DRSEMP is based on decomposition of this operation into a sequence of operations involving pairs of representing relations, one from each TOR. Most of the new representing relations carry

the same relationships as they do in their original TORs. Therefore, they are created by relatively simple operations, needed to guarantee that only tuples which do belong to objects of the new TOR will be copied to the target relations. As mentioned earlier, the JOIN creates new objects, based on the existence of objects in both operands. For example, in QUERY 7.2, if there are no female employees in EMP, then the tuples describing the dressing rooms of women will not be copied to the new TOR DRSEMP from the TOR DRESS. One relation in the new TOR establishes a new relationship in this TOR: the one that represents the VA ROOM in the new TOR. This relation establishes a direct relationship between the individual employees and their dressing rooms. In our case, this relation happens to be a result of a regular JOIN operation with regular relations. in more complicated queries, the establishment of new relationship requires more complicated activity (see later in this chapter).

The internal view of the new TOR DRSEMP is included in Table 7-8. In this internal view, note that whenever an object does not exist in one of the operands during some period, this fact is reflected in the resulting TOR. Since this is an important point, let us be specific. The first is employee 10005 that exists in the TOR EMP only since 810210. Therefore, the value of the VA ROOM for this employee is also recorded for 810210, and not for 800101, the first day in which this VA has a non-NULL value in the original TOR DRESS. An interesting case is exemplified by employe 10050. The first day at which a non-NULL value of the VA ROOM is recorded for him in DRSEMP is 800601, the first day at which this employee exists in the TOR EMP, and consequently in the TOR DRSEMP. Then, at 820508 he quits, and stops to exist in the new TOR

DRSEMP, as inherited from the TOR EMP. Therefore, not only DEPTNO and JOBCLS are recorded as NULL in the TOR DRSEMP on this day, but the VA ROOM as well. Then, at 830415 he returns, a fact that is indicated in EMP by the recording of non-NULL values for his DEPTNO and JOBCLS. These values are, of course, copied to DRSEMP. In addition, the value of the VA ROOM is evaluated for this day by interpolation based for the information in DRESS, and recorded in the new TOR DRSEMP at this day, just like DEPTNO and JOBCLS. In so doing, the data recorded for employee 10050 in DRSEMP correctly reflects the periods of time at which he does not exist.

Expressing the Key - CA JOIN by operations on the relations representing the operands, is in general:

Two relations RELA and RELB are joined, to form a new TOR RELJB. RELA has n VAs, and is represented by the relations RELA, $RELA_1$, ..., $RELA_{n+1}$. Its key is denoted by (key_attributes). RELB has m VAs, and is represented by the relations RELB, $RELB_1$, ..., $RELB_{m+1}$. The common attribute, COMM, is a CA in the first TOR and the key of the second TOR. The new TOR RELJB will be created by the following Algorithm 7.2:

1. The descriptive relation RELJB is the result of the following UNION:

   RELJB = UNION[RELA,(RELB-tuple of COMM)]

2. The relation $RELJB_1$ is the result of the following JOIN:

   JOIN $RELA_1$ WITH $RELB_1$
   INTO $RELJB_1$
   WHERE $RELA_1$.COMM=$RELB_1$.COMM

3. Let VARA be a VA in RELA, represented by $RELA_j$ in RELA and by $RELJB_k$ in the new TOR RELJB. Then, it is a result of the following JOIN:

Key	The CAs		The VAs					
EMPNO	NAME	SEX	DEPTNO		JOBCLS		ROOM	
10010	MIKE	M	800101	3	800101	4	800101	M304
			810215	2	810201	3	820512	M404
					821015	2		
10005	MARY	F	810210	2	810210	3	810210	M610
10050	DAVID	M	800601	1	800601	3	800601	M304
			820508	NULL	820508	NULL	820508	NULL
			830415	1	830415	2	830415	M404
10030	HENRY	M	800101	2	800101	3	800101	M304
			820701	3	820101	2	820512	M404
			830508	2	830304	1		
10080	ALICE	F	810101	3	810101	2	810101	M610
10025	OSCAR	M	800101	4	800101	1	800101	M304
							820512	M404
10090	SUSAN	F	800101	4	800101	4	800101	M610
					811015	3		

Table 7-8: The Internal View of the TOR DRSEMP

JOIN $RELA_j$ with $RELJB_1$
INTO $RELJB_k$
USING $RELA_j$.(key_attributes),$RELA_j$.TIME,$RELA_j$.VARA
WHERE $RELA_j$.(key_attributes)=$RELJB_1$.(key_attributes)

4. The relation representing a VA of the second TOR in the new TOR is created by the following JOIN (described for a VA VARB, represented in the second TOR by the relation $RELB_i$, and in the new TOR by the relation $RELJB_p$):

```
JOIN RELA1 WITH RELBi
INTO RELJBp
USING RELA1.(key_attributes),RELBi.TIME,
      RELBi.VARB
WHERE RELA1.COMM = RELBi.COMM
```

5. NULL values should be inserted for all VAs of objects at time points in which they do not exist, as inherited from the operands.

The steps in creating the regular relations representing the TOR DRSEMP, answering QUERY 7.2, are:

1. The descriptive relation DRSEMP is the result of the following UNION:

```
DRSEMP = UNION[EMP,(DRESS-tuple of SEX)]
```

2. The relation DRSEMP1 is the result of the following JOIN:

```
JOIN EMP1 WITH DRESS1
INTO DRSEMP1
WHERE EMP1.SEX=DRESS1.SEX
```

3. Each of the relations DRSEMP2 and DRSEMP3 is created by the following JOIN operation (described only for DRSEMP2):

```
JOIN EMP2 WITH DRSEMP1
INTO DRSEMP2
USING EMP2.EMPNO,EMP2.TIME,EMP2.DEPTNO
WHERE EMP2.EMPNO=DRSEMP1.EMPNO
```

4. The relation DRSEMP4 is the result of the following regular JOIN:

```
JOIN EMP1 WITH DRESS2
INTO DRSEMP4
USING EMP1.EMPNO,DRESS2.TIME,
                 DRESS2.ROOM
WHERE EMP1.SEX = DRESS2.SEX
```

   This JOIN establishes a relationship between EMPNO and ROOM, based upon the relationship between EMPNO and SEX in EMP1 and the relationship between SEX and ROOM in DRESS2.

5. NULL values should be inserted for all VAs of objects at time points in which they do not exist, as inherited from the operands.

As explained in Algorithm 7.2, in general, objects of either of the two operands may be dropped in the new TOR. For example, if there are no women in the TOR EMP, then the objects of the TOR DRESS, describing the dressing rooms allocated to women will be dropped by step 1 of the algorithm. In our specific example it does not happen. Therefore, the relations EMP1, EMP2 and EMP3 are practically copied from EMP to DRSEMP, and only the descriptive relation DRSEMP and the relation DRSEMP4 are new relations in the new TOR. Table 7-9 contains these relations.

The Descriptive Relation of the TOR DRSEMP

DRSEMP(ATTRIBUTE,PTYPE,LTYPE)

ATTRIBUTE	PTYPE	LTYPE
EMPNO	I4	1
NAME	C20	2
SEX	C1	2
DEPTNO	I2	3
JOBCLS	I2	3
ROOM	C4	3

The Relation Representing the VA ROOM in the TOR DRSEMP

DRSEMP4(EMPNO,TIME,ROOM)

EMPNO	TIME	ROOM
10010	800101	M304
10010	820512	M404
10005	810210	M610
10050	800601	M304
10050	820508	NULL
10050	830415	M404
10030	800101	M304
10030	820512	M404
10080	810101	M610
10025	800101	M304
10025	820512	M404
10090	800101	M610

Table 7-9: The New Relations Created for the TOR DRSEMP

This case is not as simple as the Key - Key case. However, the temporal differentiation of attributes allows us again to decompose the JOIN into a sequence of operations with the relations representing the operands that produce the new representing relations of the resulting TOR.

The data of the TOR resulting from QUERY 7.2 is actually included in the relations EMP1, EMP2, EMP3 (included in Appendix A) and DRSEMP4 (presented earlier in Table 7-9). Using these relations, we can now take a snapshot from the new TOR DRSEMP at 830101, which will then be equivalent to the result of joining the corresponding snapshots from EMP and DRESS at the same day, as presented in Table 7-7 above. This implies that the Key - CA JOIN conforms to the strong correctness criterion.

7.3. key - VA JOIN

The case Key - VA JOIN is exemplified by QUERY 7.3. This query means to complete the information about employees, already included in EMP, by adding the names and the managers of their departments, taken from the TOR DEPT.

QUERY 7.3

```
JOIN EMP WITH DEPT
INTO EMPDPT
```

The semantics of QUERY 7.3 is explained by joining snapshots taken from the operands in this JOIN. Tables 7-10 and 7-11 present snapshots taken from EMP and DEPT at 810215. Table 7-12 presents the result of their JOIN.

The Key	The CAs		The VAs	
EMPNO	NAME	SEX	DEPTNO	JOBCLS
10010	MIKE	M	2	3
10005	MARY	F	2	3
10050	DAVID	M	1	3
10030	HENRY	M	2	3
10080	ALICE	F	3	2
10025	OSCAR	M	4	1
10090	SUSAN	F	4	4

Table 7-10: A Snapshot from the TOR EMP at 810215

The resulting TOR EMPDPT consists of all possible snapshots that are results of JOIN operations with the corresponding snapshots of EMP and DEPT, as the one presented in Table 7-12.

key	a CA	a VA
DEPTNO	DEPTNM	DEPMGR
1	SALES	10050
2	PRODUCTION	10030
3	ACCOUNTING	10080
4	MANAGEMENT	10025

Table 7-11: A Snapshot from the TOR DEPT at 810215

Key	The CAs		The VAs			
EMPNO	NAME	SEX	DEPTNO	JOBCLS	DEPTNM	DEPMGR
10005	MARY	F	2	3	PRODUCTION	10030
10010	MIKE	M	2	3	PRODUCTION	10030
10025	OSCAR	M	4	1	MANAGEMENT	10025
10030	HENRY	M	2	3	PRODUCTION	10030
10050	DAVID	M	1	3	SALES	10050
10080	ALICE	F	3	2	ACCOUNTING	10080
10090	SUSAN	F	4	4	MANAGEMENT	10025

Table 7-12: The Result of Joining the Snapshots of EMP and DEPT at 810215

The implementation of this JOIN is accomplished by operations with the representing relations of the operands. The two operands have the common attribute DEPTNO which is a VA in EMP, and the Key in DEPT. Its type in EMP not only determines its own type in EMPDPT as a VA, but also the types of DEPTNM and DEPMGR as well. However, since DEPTNM is a CA in DEPT, its introduction into EMPDPT is still relatively simple. The VA DEPMGR, however,

presents a new problem caused by its type as a VA in DEPT, which in itself is a VA in EMP. Determining who is the manager of a specific employee at a particular day depends on both the department of this employee on this day, and the manager of this department on this day, and requires a special procedure. This issue is discussed in the following description of the steps to create a new TOR as a result of such a JOIN operation.

In the general case, there are two TORs, RELA and RELB, that are joined to form a new TOR RELJC. The first TOR has n VAs, and is represented by the relations RELA, $RELA_1$, ..., $RELA_{n+1}$. Its key is denoted by (key_attributes). The second TOR has m VAs, and is represented by the relations RELB, $RELB_1$, ..., $RELB_{m+1}$. The common attribute, COMM, of the two TORs is the key of RELB and a VA in RELA. It is represented by the relation $RELA_k$ in RELA. The procedure to create the new TOR RELJC is included in the following Algorithm 7.3:

1. The creation of the descriptive relation RELJC is as follows:

   RELJC = UNION(RELA,RELB)

   Change the types of all the attributes of RELB to 3 (VAs) in RELJC.

2. The relation $RELJC_k$, describing the common attribute in RELJC, is the result of the following JOIN:

   JOIN $RELA_k$ WITH $RELB_1$
   INTO $RELJC_k$
   USING $RELA_k$.(key_attributes),$RELA_k$.TIME,$RELA_k$.COMM
   WHERE $RELA_k$.COMM=$RELB_1$.COMM

3. Create a temporary relation TEMP, to contain the objects of RELJC:

   PROJECT $RELJC_k$
   ONTO (key_attributes)
   INTO TEMP

Every VA, VARA, in the TOR RELA (represented by the relation $RELA_1$) will be copied to the relation $RELJC_q$ in the new TOR by the following JOIN:

```
JOIN RELA1 with TEMP
INTO RELJCq
WHERE RELA1.(key_attributes)=TEMP.(key_attributes)
```

4. The CAs relation $RELJC_1$ is created by the following JOIN:

```
JOIN RELA1 WITH TEMP
INTO RELJC1
WHERE RELA1.(key_attributes)=TEMP.(key_attributes)
```

5. Every CA, say CONST, of the TOR RELB, becomes a VA in RELJC, and is represented by a separate relation, say $RELJC_i$ in it, created by the following JOIN:

```
JOIN RELJCk WITH RELB1
INTO RELJCi
USING RELJCk.(key_attributes),RELJCk.TIME,
      RELB1.CONST
WHERE RELJCk.COMM=RELB1.COMM
```

This operation replaces each value of COMM in $RELJC_k$ by its corresponding value of CONST in the TOR RELB, and records the results in the relation $RELJC_i$.

6. Each VA in the TOR RELB will be a VA in the new TOR too. Assume that one of these VAs is VAR, it is represented by the relation $RELB_j$ in RELB, and will be represented by the relation $RELJC_p$ in the new TOR. This relation describes the values of this VA as functions of TIME for each of the objects <u>of the first TOR</u>, while originally, in $RELB_j$, it describes these values for each of the objects <u>of the second TOR</u>. Therefore, the relation $RELJC_p$ is produced by a merge of $RELJC_k$ and $RELB_j$, that cannot be translated to a relational algebra operation. Rather, it is done by scanning both relations simultaneously, identifying the VAR value of each of the first TOR's objects at all the time points included in its tuples in $RELJC_k$, and recording this information in $RELJC_p$. In addition, this procedure creates also the additional necessary tuples of $RELJC_p$, reflecting those objects of the first TOR that are affected by changes in VAR values, as described in the relation $RELB_j$.

7. NULL values should be inserted for all VAs of objects in RELJC at time points in which they do not exist, as inherited from the operands.

This procedure, applied to QUERY 7.3, creates the new TOR EMPDPT as follows:

1. The creation of the descriptive relation EMPDPT is as follows:

```
EMPDPT = UNION(EMP,DEPT)
```

   With the types of all the attributes of DEPT changed to 3 (VAs) in EMPDPT.

2. The relation EMPDPT2 , describing the VA DEPT in EMPDPT, is obtained by the following JOIN:

```
JOIN EMP2 WITH DEPT1
INTO EMPDPT2
USING EMP2.EMPNO,EMP2.TIME,EMP2.DEPTNO
WHERE EMP2.DEPTNO=DEPT1.DEPTNO
```

3. Create a temporary relation TEMP, to contain the objects of EMPDPT:

```
PROJECT EMPDPT2
ONTO EMPNO
INTO TEMP
```

   The relation EMPDPT3, representing JOBCLS in EMPDPT is a result of the following JOIN:

```
JOIN EMP3 WITH TEMP
INTO EMPDPT3
WHERE EMP3.EMPNO=TEMP.EMPNO
```

4. The relation EMPDPT1, describing the CAs in the new TOR is created by the following JOIN:

```
JOIN EMP1 WITH TEMP
INTO EMPDPT1
WHERE EMP1.EMPNO=TEMP.EMPNO
```

5. EMPDPT4 describes the VA DEPTNM that, being a CA in DEPT, is determined uniquely by DEPTNO. Therefore, the creation of EMPDPT4 is achieved by:

```
JOIN EMPDPT2 WITH DEPT1
INTO EMPDPT4
USING EMPDPT2.EMPNO,EMPDPT2.TIME,
```

```
DEPT1.DEPTNM
WHERE EMPDPT2.DEPTNO=DEPT1.DEPTNO
```

6. The relation EMPDPT5 describes the values of DEPMGR as functions of TIME for each employee. It is created by scanning EMPDPT2 and DEPT2 simultaneously, identifying the manager of each employee at all the time points included in his tuples in EMPDPT2, and recording this information in EMPDPT5. In addition, this procedure creates the additional necessary tuples of EMPDPT5, reflecting the employees affected by changes in departments managers, as described in DEPT2.

7. NULL values should be recorded for all VAs of objects at time points in which they do not exist. E.g., employee 10050 in the TOR EMP does not exist during the period 820508 - 830414. Therefore, in the resulting TOR EMPDPT, all the VAs of this object should be NULL during this period. The VAs inherited from EMP are clearly NULL, and NULL values are inserted in the relations representing the VAs DEPTNM and DEPMGR in the new TOR, VAs that are inherited from the TOR DEPT.

The new relation EMPDPT5 is presented in Table 7-13. The presentation of all other relations that represent the new TOR EMPDPT is omitted, since they are either existing relations, or easily derivable from existing ones. In this specific example all the objects of the two operands are included in the new TOR, but as explained before, in general some of them may be ignored. For example, if no employee ever worked for department 3, then the information about this department in DEPT will be ignored.

This query illustrates the complexities involved in performing the JOIN operation, and the dependency of this operation upon the types of the common attributes in the participating TORs. Nevertheless, decomposing the JOIN into a sequence of a fairly isolated operations on the representing relations of the operands, helps to reduce the complexity of this JOIN. Basically, in a VA - Key JOIN, the objects of the new TOR are of the same kind as those of the TOR in which the common attribute is a VA. Objects that appear only in one of them (i.e., they do not have matching objects in the other TOR) are omitted from the result, and therefore the relations representing them in the new TOR

EMPDPT5(EMPNO,TIME,DEPMGR)

EMPNO	TIME	DEPMGR
10005	810210	10030
10005	820701	10005
10010	800101	10010
10010	810215	10030
10010	820701	10005
10025	800101	10025
10030	800101	10030
10030	820701	10080
10030	830508	10005
10050	800601	10050
10050	820508	NULL
10050	830415	10050
10080	810101	10010
10080	810215	10080
10090	800101	10025

Table 7-13: The Relation Representing the Managers of All Employees at All Time Points

cannot simply be copied from those of one TOR, but must be created by regular JOIN operations that select only the proper objects.

The objects of the second TOR (in which the common attribute is the key) just add information to the corresponding objects of the first TOR. In addition, all the second TOR's attributes become VAs in the new TOR. Therefore, new relationships should be established between them and the key of the new TOR (which is inherited from the first TOR). This issue is relatively

simple when dealing with the CAs of the second TOR, and is resolved by appropriate JOIN operations. Transfering a VA from the second TOR to the new TOR is more complicated. It requires a special operation that infers the temporal path of this VA in the objects of the new TOR, derived from its variation in the second TOR and from the variation of the second TOR's key as a VA in the first TOR.

The VA - Key operation, as defined here, satisfies the strong correctness criterion. Taking a snapshot from the new TOR EMPDPT at 810215, produces a relation that is identical to the result of joining snapshots from EMP and DEPT at this day (Table 7-12 above).

7.4. CA - CA JOIN

This case is presented by using the TORs EMP and UNIONS in QUERY 7.4 below. The TOR UNIONS contains information about unions that are present in the company and their offices (there are "currently" three unions, two for men only and one for women only). The content of this TOR is included in Appendix A. It is assumed that every employee is a member in all the unions that correspond to his/her sex.

QUERY 7.4

```
JOIN UNION WITH EMP
INTO UNEMP
```

The purpose of QUERY 7.4 is to associate each individual in the company with the unions of which he or she is a member, and present the combined

information of the employees and their unions in one TOR. The meaning of QUERY 7.4 is illustrated by taking snapshots from the operands at an arbitrary day, and joining them to produce a relation that is one of many of which the new TOR conceptually consists. The two snapshots (at 810215) are presented in Tables 7-14 and 7-15. Their JOIN is presented in Table 7-16.

Key	The CAs		The VAs	
EMPNO	NAME	SEX	DEPTNO	JOBCLS
10005	MARY	F	2	3
10010	MIKE	M	2	3
10025	OSCAR	M	4	1
10030	HENRY	M	2	3
10050	DAVID	M	1	3
10080	ALICE	F	3	2
10090	SUSAN	F	4	4

Table 7-14: A Snapshot from the TOR EMP at 810215

Key	CA	VA
UNION	SEX	OFFICE
ALPHA	M	M101
BETA	F	M102
GAMA	M	W203

Table 7-15: A Snapshot from the TOR UNIONS at 810215

The new TOR UNEMP conceptually consists of all possible snapshots like the one represented in Table 7-16. This TOR contains new type of object,

The Key		The CAs		The VAs		
UNION	EMPNO	SEX	NAME	OFFICE	DEPTNO	JOBCLS
ALPHA	10010	M	MIKE	M101	2	3
ALPHA	10025	M	OSCAR	M101	4	1
ALPHA	10030	M	HENRY	M101	2	3
ALPHA	10050	M	DAVID	M101	1	3
BETA	10005	F	MARY	M102	2	3
BETA	10080	F	ALICE	M102	3	2
BETA	10090	F	SUSAN	M102	4	4
GAMA	10010	M	MIKE	W203	2	3
GAMA	10025	M	OSCAR	W203	4	1
GAMA	10030	M	HENRY	W203	2	3
GAMA	10050	M	DAVID	W203	1	3

Table 7-16: The Result of Joining Snapshots of EMP and UNIONS at 810215

namely "union members", i.e., all combinations of UNIONs and EMPNOs that exist. The conceptual construction of this JOIN is indeed similar to previous JOIN operations analyzed in this chapter. Its implementation is symmetric (like the Key - Key JOIN above), and the two operands are treated the same. The new key is a combination of the two operands' keys. The common attribute is a CA that depends upon each of them in its TOR, and therefore the new TOR should reflect this dependency, by having this combination of the keys as its key. Consequently, all other attributes maintain their original types, since their dependency and variation do not change in their transition to the new TOR. The internal view of the new TOR UNEMP is presented in Table 7-17.

The Key		The CAs		The VAs					
UNION	EMPNO	SEX	NAME	OFFICE		DEPTNO		JOBCLS	
ALPHA	10010	M	MIKE	800501	M101	800501	3	800501	4
				810615	N503	810215	2	810201	3
								821015	2
ALPHA	10050	M	DAVID	800601	M101	800601	1	800601	3
				810615	N503	820508	NULL	820508	NULL
				820508	NULL	830415	1	830415	2
				830415	N503				
ALPHA	10030	M	HENRY	800501	M101	800501	2	800501	3
				810615	N503	820701	3	820101	2
						830508	2	830304	1
ALPHA	10025	M	OSCAR	800501	M101	800501	4	800501	1
				810615	N503				
BETA	10005	F	MARY	810210	M102	810210	2	810210	3
				810620	N505				
BETA	10080	F	ALICE	810101	M102	810101	3	810101	2
				810620	N505				
BETA	10090	F	SUSAN	800601	M102	800601	4	800601	4
				810620	N505			811015	3

The Key		The CAs		The VAs					
UNION	EMPNO	SEX	NAME	OFFICE		DEPTNO		JOBCLS	
GAMA	10010	M	MIKE	810210	W203	810210	3	810210	4
						810215	2	810201	3
								821015	2
GAMA	10050	M	DAVID	810210	W203	810210	1	810210	3
				820508	NULL	820508	NULL	820508	NULL
				830415	W203	830415	1	830415	2
GAMA	10030	M	HENRY	810210	W203	810210	2	810210	3
						820701	3	820101	2
						830508	2	830304	1
GAMA	10025	M	OSCAR	810210	W203	810210	4	810210	1

Table 7-17: The Internal View of the TOR UNEMP

The general CA - CA JOIN joins two TORs, RELA and RELB, to form the TOR RELJD. The TOR RELA has n VAs, and is represented by the relations RELA, $RELA_1$, ..., $RELA_{n+1}$. Its key is denoted by (key_A). The TOR RELB has m VAs, and is represented by the relations RELB, $RELB_1$, ..., $RELB_{m+1}$. Its key is denoted by (key_B). The two TORs have one common attribute, COMM, which is a CA in both of them. The resulting TOR RELJD is created through the following Algorithm 7.4:

1. The descriptive relation RELJD is the result of the following UNION:

```
RELJD = UNION(RELA,RELB)
```

2. The relation $RELJD_1$ is the result of the following JOIN:

```
JOIN RELA1 WITH RELB1
INTO RELJD1
WHERE RELB1.COMM = RELA1.COMM
```

3. A VA in the first TOR, say VARA represented by $RELA_i$, is represented by a relation, say $RELJD_j$, in RELJD, created by the following JOIN:

```
JOIN RELJD1 WITH RELAi
INTO RELJDj
USING RELAi.(key_A),RELJD1.(key_B),
      RELAi.TIME,RELAi.VARA
WHERE RELJD1.(key_A) = RELAi.(key_A)
```

4. Similarly, a VA VARB in RELB, represented by $RELB_k$, will be represented by $RELJD_1$, created by the following JOIN:

```
JOIN RELJD1 WITH RELBk
INTO RELJD1
USING RELJD1.(key_A),RELBk.(key_B),
      RELBk.TIME,RELBk.VARB
WHERE RELJD1.(key_B) = RELBk.(key_B)
```

5. NULL values are recorded for all VAs of objects at time points in which they do not exist, as inherited from the operands, including periods before the first explicitly recorded values in the operands for all their objects.

The new TOR UNEMP, answering QUERY 7.4, is created by the following procedure:

1. The relation UNEMP is the result of the following UNION:

```
UNEMP = UNION(EMP,UNIONS)
```

2. The relation UNEMP1 is the result of the following regular JOIN:

```
JOIN EMP1 WITH UNIONS1
INTO UNEMP1
USING UNIONS.UNION,EMP.EMPNO,
      EMP.SEX,EMP.NAME
WHERE UNIONS1.SEX = EMP1.SEX
```

3. The relation UNEMP2 is the result of the following regular JOIN:

```
JOIN UNEMP1 WITH UNIONS2
INTO UNEMP2
USING UNIONS2.UNION,UNEMP1.EMPNO,
      UNIONS2.TIME,UNIONS2.OFFICE
WHERE UNEMP1.UNION = UNIONS2.UNION
```

The relation UNEMP3 is the result of the following regular JOIN:

```
JOIN UNEMP1 WITH EMP2
INTO UNEMP3
USING UNEMP1.UNION,EMP2.EMPNO,
      EMP2.TIME,EMP2.DEPTNO
WHERE UNEMP1.EMPNO = EMP2.EMPNO
```

4. The last relation UNEMP4 is the result of a regular JOIN, similar to the previous one, where EMP2 is replaced by EMP3 and DEPTNO is replaced by JOBCLS.

5. NULL values are recorded for all VAs of objects at time points in which these objects do not exist. In QUERY 7.4, such a situation happens with employee 10050. NULL values were inserted for the VA OFFICE of all the objects in the new TOR UNEMP (Table 7-17 above) which contain EMPNO=10050 as one of their components, at the time points implied by NULL values in both VAs DEPTNO and JOBCLS. Also, non-NULL values (obtained by interpolation in the TOR UNIONS) were assigned at a later time points to the VA OFFICE, as implied by DEPTNO and JOBCLS values of the object 10050 in EMP. In addition, the first non-NULL values for all VAs of all objects are recorded for the latest time point of the two that appear in DEPTNO and JOBCLS (in EMP for the EMPNO component of the objects) on one hand, and in OFFICE (in UNIONS for the UNION component of the objects) on the other hand. This makes the information in the new TOR consistent with the way of indicating periods of time at which objects do not exist, by NULL values assigned to all their VAs during these periods. In so doing, these objects have implicit NULL values during the period of time before both of their components (EMPNO in EMP, and UNION in UNIONS) first exist in their original TORs.

From implementational point of view, the CA - CA JOIN is a symmetric case. This symmetry is reflected in the underlying procedure: the new key is the union of the original keys; all the attributes maintain their original

types; and the transformation of each attribute from its original TOR to the new one depends only on its type, and not on the identity of the operand to which it belongs. As could be seen through inspection of Tables 7-17 and 7-16, this JOIN could be defined to satisfy the strong correctness criterion.

7.5. CA - VA JOIN

The TORs EMP and PHONES are used in presenting this case. The TOR PHONES describes the various telephones used by the various departments in our organization. Its key is the telephone number; it has the CA DEPTNO, indicating the department to which it is assigned, and the VA LINES indicating the number of lines allocated to each number at each point of time. The content of the TOR PHONES is included in Appendix A.

QUERY 7.5 below demonstrates the CA - VA JOIN. It associates the individuals in the organization with the telephone numbers through which they can be reached, according to their departments.

QUERY 7.5

```
JOIN PHONES WITH EMP
INTO PNEMP
```

We illustrate the meaning of QUERY 7.5 by joining two snapshots taken from the operands at an arbitrary time point, say 830101. These snapshots are presented in Tables 7-18 and 7-19. The result of their JOIN is presented in Table 7-20. The new TOR PNEMP conceptually consists of such relations at all

time points. Therefore, this TOR describes the associations between employees and departmental telephone numbers at all time points.

The Key	The CAs		The VAs	
EMPNO	NAME	SEX	DEPTNO	JOBCLS
10010	MIKE	M	2	2
10005	MARY	F	2	3
10030	HENRY	M	3	2
10080	ALICE	F	3	2
10025	OSCAR	M	4	1
10090	SUSAN	F	4	3

Table 7-18: A Snapshot from the TOR EMP at 830101

The Key	a CA	a VA
PHONE	DEPTNO	LINES
2856010	1	1
2856040	2	3
2856090	2	2
2856110	3	1
2856000	4	1
2856100	4	1

Table 7-19: A Snapshot from the TOR PHONES at 830101

As shown in Table 7-20, this JOIN creates new objects identified by the combination of the two operands' keys. The reason for this is that the common attribute is a non-key attribute in both operands. Since this attribute depends on both keys (as expressed in the two TORs), this dependency should be

The Key		The CAs		The VAs		
EMPNO	PHONE	NAME	SEX	DEPTNO	JOBCLS	LINES
10005	2856040	MARY	F	2	3	3
10005	2856090	MARY	F	2	3	2
10010	2856040	MIKE	M	2	2	3
10010	2856090	MIKE	M	2	2	2
10025	2856000	OSCAR	M	4	1	1
10025	2856100	OSCAR	M	4	1	1
10030	2856110	HENRY	M	3	2	1
10080	2856110	ALICE	F	3	2	1
10090	2856000	SUSAN	F	4	3	1
10090	2856100	SUSAN	F	4	3	1

Table 7-20: Joining the Snapshots from EMP and PHONES at 830101 reflected in the new TOR, by having both keys combining the new key (this property is inherited from the regular relational model). The common attribute DEPTNO will be a VA in the new TOR, as it is in EMP, and not a CA, its type in PHONES. The internal view of the new TOR PNEMP is presented in Table 7-21.

The Key		The CAs		The VAs		
EMPNO	PHONE	NAME	SEX	DEPTNO	JOBCLS	LINES
10010	2856110	MIKE	M	800101 3 810215 NULL	800101 4 810201 NULL	800101 1 810215 NULL
10010	2856040	MIKE	M	810215 2	810215 3 821015 2	810215 1 810304 3
10010	2856090	MIKE	M	810215 2	810215 3 821015 2	810215 2
10005	2856040	MARY	F	810210 2	810210 3	810210 1 810304 3
10005	2856090	MARY	F	811001 2	811001 3	811001 2
10050	2856010	DAVID	M	800601 1 820508 NULL 830415 1	800601 3 820508 NULL 830415 2	800601 1 820508 NULL 830415 1
10030	2856040	HENRY	M	800101 2 820701 NULL 830508 2	800101 3 820101 2 820701 NULL 830508 1	800101 1 810304 3 820701 NULL 830508 3
10030	2856090	HENRY	M	811001 2 820701 NULL 830508 2	811001 3 820101 2 820701 NULL 830508 1	811001 2 820701 NULL 830508 2
10030	2856110	HENRY	M	820701 3 830508 NULL	820701 2 830304 1 830508 NULL	820701 1 830508 NULL

The Key		The CAs		The VAs					
EMPNO	PHONE	NAME	SEX	DEPTNO		JOBCLS		LINES	
10080	2856110	ALICE	F	810101	3	810101	2	810101	1
10025	2856000	OSCAR	M	800101	4	800101	1	800101	1
10025	2856100	OSCAR	M	800401	4	800401	1	800401	1
10090	2856000	SUSAN	F	800101	4	800101	4	800101	1
						811015	3		
10090	2856100	SUSAN	F	800401	4	800401	4	800401	1
						811015	3		

Table 7-21: The Internal View of the TOR PNEMP

Let us, for instance, inspect the object EMPNO=10030 and PHONE=2856090 in Table 7-21. The CAs of this object are those of the TOR EMP. Specifically, NAME=HENRY and SEX=M are inherited from employee 10030. The determination of the data in its VAs is more complicated. All of them should be NULL at time points in which this object does not exist (i.e., employee 10030 is not in the department to which the telephone 2856090 is assigned). At time points in which this object does exist, its VAs should inherit their values from their original TORs. Specifically, PHONE=2856090 is assigned to DEPTNO=2 starting at 811001 (we learn this from the fact that the VA LINE is assigned its first non-NULL value for PHONE=2856090 in the TOR PHONES at 811001). Therefore, only the period in which EMPNO=10030 was assigned to department 2, <u>after 811001</u> is relevant to the object EMPNO=10030 and PHONE=2856090. All VAs for this object in the new TOR (DEPTNO, JOBCLS and LINES) inherit their values at this day from the original TORs. At 820101, a change occurred in the value of

JOBCLS in EMP for object 10030. The day 820101 is included in the period at which the object EMPNO=10030 and PHONE=2856090 exists, and therefore this change is recorded in the new TOR. Then, at 820701 employee 10030 leaves department 2, implying the termination of the object EMPNO=10030 and PHONE=2856090 in the new TOR. Therefore, NULL values are recorded for DEPTNO, JOBCLS and LINES of this object in the new TOR. Then, at 830508 employee 10030 returns to department 2. The telephone 2856090 is still assigned to this department at this day (the value of the VA LINE in the TOR PHONES is non-NULL for this telephone at this day). Therefore, the values of JOBCLS and LINES (in addition to DEPTNO), derived from the operands at this day are recorded for the object EMPNO=10030 and PHONE=2856090 in the new TOR. According to the data of employee 10030 in EMP and the data of telephone 2856090 in PHONES, no further changes are recorded in the VAs of the object EMPNO=10030 and PHONE=2856090 in the new TOR.

The problems involved in formulating this JOIN are similar to those encountered earlier in the Key - VA JOIN, but they are nevertheless much deeper. Here we have a CA instead of the key, and therefore the key of the new TOR UNEMP contains the keys of both TORs participating in the JOIN. As in the result of QUERY 7.3, the common attribute DEPTNO will be a VA in the resulting TOR, derived from its type in EMP. Yet, all other attributes will maintain their original types. The creation of the VA LINES in the resulting TOR PNEMP is similar to the creation of the VA DEPMGR in the resulting TOR of QUERY 7.3. In the TOR UNEMP, the VA LINES is associated not only with PHONE, but with the combination of PHONE and EMPNO, and therefore it is affected by

its original values in the TOR PHONES together with the values of DEPTNO in the TOR EMP. However, its dependency on DEPTNO in the TOR PHONES is not an explicit one like the dependency of an attribute on its key, and therefore the procedure to determine the values of LINES in the new TOR is more complicated than the one used in QUERY 7.3, to relate employees to their managers.

Let us present the general case of CA - VA JOIN. Two TORs, RELA and RELB are joined to form a new TOR RELJE. The TOR RELA has n VAs, and is represented by the relations RELA, $RELA_1$, ..., $RELA_{n+1}$. Its key is denoted by (key_A). The TOR RELB has m VAs, and is represented by the relations: RELB, $RELB_1$, ..., $RELB_{m+1}$. Its key is denoted by (key_B). The two TORs has one attribute, COMM, in common. This attribute is a CA in RELA and a VA in RELB. It is stored in the relation $RELB_i$ in RELB, and in the relation $RELA_1$ in RELA. This attribute will be represented by the relation $RELJE_k$ in the new TOR RELJE. The steps according to which the new TOR RELJE is created are included in the following Algorithm 7.5:

1. The relation RELJE is the result of the following UNION:

   RELJE = UNION[(RELA-tuple of COMM),RELB]

2. The relation $RELJE_1$ is created by the following JOIN operations:

```
JOIN RELA1 WITH RELBi
INTO TEMP
USING RELA1.(key_A),RELBi.(key_B)
WHERE RELBi.COMM = RELA1.COMM

JOIN RELA1 WITH TEMP
INTO TEMP1
WHERE RELA1.(key_A)=TEMP.(key_A)

JOIN RELB1 WITH TEMP1
INTO RELJE1
WHERE RELB1.(key_B)=TEMP.(key_B)
```

3. The relation $RELJE_k$ describes the VA COMM in RELJE. It is created by the following operations:

```
JOIN TEMP WITH RELB_i
INTO RELJE_k
WHERE TEMP.(key_B)=RELB_i.(key_B)
```

4. The following steps create the relation $RELJE_1$, representing a VA VARA in RELJE. It is represented by $RELA_j$ in RELA.

```
JOIN TEMP WITH RELA_j
INTO RELJE_1
WHERE TEMP.(key_A)=RELA_j.(key_A)
```

5. Assume that the VA VARB is represented in RELB by $RELB_p$. It will be represented by $RELJE_q$ in RELJE, through to the following JOIN:

```
JOIN TEMP WITH RELB_p
INTO RELJE_q
WHERE TEMP.(key_B)=RELB_p.(key_B)
```

6. NULL values are recorded in all VAs of objects at time points in which they do not exist, based on the NULL values recorded in the VAs of objects in the operands, and on the values of the common attribute in particular.

The procedure to create the TOR PNEMP, answering QUERY 7.5, is therefore:

1. The relation PNEMP is the result of the following UNION:

```
PNEMP = UNION[EMP,(PHONES-tuple of DEPTNO)]
```

2. The relation PNEMP1 is the result of the following JOIN operations:

```
JOIN PHONES1 WITH EMP2
INTO TEMP
USING EMP2.EMPNO,PHONES1.PHONE
WHERE EMP2.DEPTNO = PHONES1.DEPTNO

JOIN EMP1 WITH TEMP
INTO PNEMP1
USING EMP1.EMPNO,TEMP.PHONE,
      EMP1.NAME,EMP1.SEX
WHERE EMP1.EMPNO = TEMP.EMPNO
```

3. The relation PNEMP2 describes the VA DEPTNO in PNEMP. It is created by the following operations:

```
JOIN TEMP WITH EMP2
INTO PNEMP2
USING TEMP.EMPNO,TEMP.PHONE
      EMP2.TIME,EMP2.DEPTNO
WHERE TEMP.EMPNO = EMP2.EMPNO
```

4. The relation PNEMP3, describing the VA JOBCLS in the TOR PNEMP will be created by the following JOIN:

```
JOIN TEMP WITH EMP3
INTO PNEMP3
USING TEMP.EMPNO,TEMP.PHONE,
      EMP3.TIME,EMP3.JOBCLS
WHERE TEMP.EMPNO = EMP3.EMPNO
```

5. The relation PNEMP4, describing LINES in PNEMP, is created by the following JOIN:

```
JOIN TEMP WITH PHONES2
INTO PNEMP4
USING TEMP.EMPNO,TEMP.PHONE,
      PHONES2.TIME,PHONES2.LINES
WHERE TEMP.PHONE = PHONES2.PHONE
```

6. NULL values are recorded in all the VAs of objects at time points in which they do not exist, based upon the non-existence of objects in the operands, and upon the values of the common attribute in them.

The CA - VA JOIN, like all previous JOIN operation discussed in this chapter, satisfies the strong correctness criterion. For instance, Table 7-22 contains a snapshot taken from the new TOR PNEMP at 830101. Comparing Table 7-22 and Table 7-20 reveals that all the tuples that do not contain NULL values for all their VAs are identical in both of them. In addition, Table 7-22 contains some tuples in which all VAs are NULL. These tuples represent objects that do not exist at 830101 (in our case, employees and telephone numbers that are not associated with the same department at 830101). Therefore, these two tables are equivalent.

The Key		The CAs		The VAs		
EMPNO	PHONE	NAME	SEX	DEPTNO	JOBCLS	LINES
10005	2856040	MARY	F	2	3	3
10005	2856090	MARY	F	2	2	2
10010	2856040	MIKE	M	2	2	3
10010	2856090	MIKE	M	2	2	2
10010	2856110	MIKE	M	NULL	NULL	NULL
10025	2856000	OSCAR	M	4	1	1
10025	2856100	OSCAR	M	4	1	1
10030	2856040	HENRY	M	NULL	NULL	NULL
10030	2856090	HENRY	M	NULL	NULL	NULL
10030	2856110	HENRY	M	3	2	1
10050	2856010	DAVID	M	NULL	NULL	NULL
10080	2856110	ALICE	F	3	2	1
10090	2856000	SUSAN	F	4	3	1
10090	2856100	SUSAN	F	4	3	1

Table 7-22: A Snapshot From The TOR PNEMP at 830101

7.6. VA - VA JOIN

This case is presented using a JOIN operation with the TORs EMP and PROJECTS. The TOR PROJECTS describes the various projects carried out by the company. The key of this TOR is PROJNO, and the name of the project (PROJNM) is obviously a CA. The VA COST describes the costs estimates of each project, as they develop over time. The VA DEPTNO specifies the department which is in charge of each project at each point of time.

QUERY 7.6 illustrates a VA - VA JOIN. It associates the various employees with the various projects carried out by the organization, through the departments to which the employees belong and in which the projects are

handled. We associate each employee with all of the projects in his/her department.

QUERY 7.6

```
JOIN PROJECTS WITH EMP
INTO  PRJEMP
```

The meaning of QUERY 7.6 is illustrated by snapshots of the two operands. They are presented in Tables 7-23 and 7-24. The result of their JOIN is presented in Table 7-25.

The Key	The CAs		The VAs	
EMPNO	NAME	SEX	DEPTNO	JOBCLS
10010	MIKE	M	2	2
10005	MARY	F	2	3
10050	DAVID	M	1	2
10030	HENRY	M	2	1
10080	ALICE	F	3	2
10025	OSCAR	M	4	1
10090	SUSAN	F	4	3

Table 7-23: A Snapshot from the TOR EMP at 831215

Table 7-25 contains all the combinations of employees and projects that are associated at 831215. The new TOR PRJEMP conceptually consists of such relations at all time points. The employee component of each object in the new TOR "contributes" all the data of this employee (NAME, SEX, DEPTNO and

Key	a CA	The VAs	
PROJNO	PROJNM	COST	DEPTNO
1000	MDA	9000	1
1010	AI	22000	3
1020	TDMS	18000	2

Table 7-24: A Snapshot from the TOR PROJECTS at 831215

The Key		The CAs			The VAs		
PROJNO	EMPNO	PROJNM	NAME	SEX	COST	DEPTNO	JOBCLS
1000	10050	MDA	DAVID	M	9000	1	2
1010	10080	AI	ALICE	F	22000	3	2
1020	10010	TDMS	MIKE	M	18000	2	2
1020	10030	TDMS	HENRY	M	18000	2	1
1020	10005	TDMS	MARY	F	18000	2	3

Table 7-25: Joining The Snapshots of EMP and PROJECTS at 831215

JOBCLS), while the project component "contributes" the data about the project (again, its DEPTNO, and its COST). DEPTNO, the common attribute, appears, of course, only once for each object.

From an implementation point of view, the VA - VA JOIN is a very complicated operation, since the two TORs are combined through VAs. On the other hand, this is a symmetric operation, since the common attribute has the same type in the two operands. The key of the new TOR is the combination of the two operands' keys, since the common attribute functionally depends on

them in the two TORs, and the resulting TOR should reflect these dependencies. Once the keys maintain their original types in the new TOR, all other attributes maintain their original types too. This property is inherited from the regular relational model.

In order to follow this JOIN operation, let us take a closer look at the "story" of employee 10030 with project 1000. This project started at 800805 under the responsibility of department 4, and employee 10030 had nothing to do with it. However, at 801220 department 2 took the responsibility for this project, and employee 10030, who worked for this department on this day, became involved with this project. At 811015, the project moved to department 3, and therefore the association between employee 10030 and this project terminated. AT 820701, employee 10030 moved to department 3, and became again involved with project 1000. AT 821220 the project moved to department 1, and from this day on, there was no association between employee 10030 and project 1000. The new TOR PRJEMP should reflect such "stories" with full accuracy.

The Key		a CA	The VAs					
PROJNO	EMPNO	PROJNM	COST		DEPTNO		JOBCLS	
1000	10030	MDA	801220	5000	801220	2	801220	3
			811015	NULL	811015	NULL	811015	NULL
			820701	9000	820701	3	820701	2
			821210	NULL	821210	NULL	821210	NULL

Table 7-26: The Object EMPNO=10030 and PROJNO=1000 in the Internal View of the TOR PRJEMP

Table 7-26 presents the information corresponding to the "story" of the association between employee 10030 and project 1000, as described above, omitting - for presentation purposes - the CAs NAME and SEX. Note that at all time points in which there is no association between employee 10030 and project 1000, all the VAs of this object are NULL, while in those time points in which this object exists, its VAs are the values derived from the operands.

The general procedure for handling a VA - VA JOIN is as follows: There are two TORs, RELA and RELB, that are joined to form a new TOR RELJF. The first TOR has n VAs, and is represented by the relations RELA, $RELA_1$, ..., $RELA_{n+1}$. Its key is denoted by (key_A). The second TOR has m VAs, and is represented by the relations RELB, $RELB_1$, ..., $RELB_{m+1}$. Its key is denoted by (key_B). The two TOR have one common attribute, VAR, which is a VA in both of them. It is represented by the relation $RELA_i$ in one TOR, and by the relation $RELB_j$ in the other. The steps to create the relations representing the new TOR RELJF are included in the following Algorithm 7.6:

1. The relation RELJF is the result of the following UNION:

   RELJF = UNION(RELA,RELB)

2. The new objects are included in the relation TEMP, created by the following JOIN:

   JOIN $RELA_i$ WITH $RELB_j$
   INTO TEMP
   USING $RELA_i$.(key_A),$RELB_j$.(key_B)
   WHERE $RELA_i$.VAR = $RELB_j$.VAR

3. The relation $RELJF_1$ is the result of the following JOIN operations:

   JOIN TEMP WITH $RELA_1$
   INTO TEMP1
   WHERE TEMP.(key_A)=$RELA_1$.(key_A)

JOIN TEMP1 WITH $RELB_1$
INTO $RELJF_1$
WHERE TEMP1.(key_B)=$RELB_1$.(key_B)

4. The following JOIN creates the relation $RELJF_1$ representing the non-common VA VARA of RELA (represented by $RELA_k$ in RELA):

JOIN TEMP WITH $RELA_k$
INTO $RELJF_1$
WHERE TEMP.(key_A) = $RELA_k$.(key_A)

5. The following JOIN creates the relation $RELJF_q$, representing the non-common VA VARB of RELB (represented by $RELB_p$ in RELB):

JOIN TEMP WITH $RELB_p$
INTO $RELJF_q$
WHERE TEMP.(key_B)=$RELB_p$.(key_B)

6. As mentioned earlier, the creation of the relation that describes the common attribute VAR in the new TOR, requires special procedure, similar to those already introduced in queries 6.2 and 6.6. Assume that this relation is $RELJF_r$. This procedure contains a simultaneous scanning of the relations $RELA_i$ and $RELB_j$, in order to identify for each object (consisting of a combination of (key_A) and (key_B)) the precise periods of time in which VAR is simultaneously associated with the object (key_A) in RELA and with the object (key_B) in RELB. At these periods, the proper values of VAR are recorded. At others, NULL values are recorded.

7. NULL values should be recorded in all VAs of objects at time points in which they do not exist, based upon the existence of objects in the operands, and upon the values of VAR in them.

The same procedure, applied to creating PRJEMP, is as follows:

1. The relation PRJEMP is the result of the following UNION:

PRJEMP = UNION(EMP,PROJECTS)

2. The temporary relation TEMP, created by the following JOIN, contains the objects of the new TOR.

JOIN PROJECTS3 WITH EMP2
INTO TEMP
USING PROJECTS3.PROJNO,EMP2.EMPNO
WHERE EMP2.DEPTNO = PROJECTS3.DEPTNO

3. The relation PRJEMP1 is the result of the following JOIN operations:

```
JOIN TEMP WITH EMP1
INTO TEMP1
USING TEMP.PROJNO,TEMP.EMPNO,
      EMP1.NAME,EMP1.SEX
WHERE TEMP.EMPNO = EMP1.EMPNO

JOIN TEMP1 WITH PROJECTS1
INTO PRJEMP1
USING TEMP1.PROJNO,TEMP1.EMPNO,
      PROJECTS1.PROJNM,TEMP1.NAME,
      TEMP1.SEX
WHERE TEMP1.PROJNO = PROJECTS1.PROJNO
```

4. The relation PRJEMP2 describes the VA COST in the new TOR PRJEMP. It is the result of the following JOIN:

```
JOIN TEMP WITH PROJECTS2
INTO PRJEMP2
USING TEMP.PROJNO,TEMP.EMPNO,
      PROJECTS2.TIME,PROJECTS2.COST
WHERE TEMP.PROJNO = PROJECTS2.PROJNO
```

5. PRJEMP4 is created by the following JOIN:

```
JOIN TEMP WITH EMP3
INTO PRJEMP4
USING TEMP.PROJNO,TEMP.EMPNO,
      EMP3.TIME,EMP3.JOBCLS
WHERE TEMP.EMPNO = EMP3.EMPNO
```

6. The relation PRJEMP3 describes the VA DEPTNO in the new TOR PRJEMP. Since DEPTNO is the common attribute in this JOIN, the creation of PRJEMP3 requires a special procedure, similar to those, already introduced in QUERY 7.3 and QUERY 7.5. This procedure contains a simultaneous scanning of the relations EMP2 and PROJECTS3, in order to identify for each object (consisting of a combination of PROJNO and EMPNO) the precise periods of time in which the project is under the responsibility of each department, concurrent with the employee being in the department. At these periods, the proper values of DEPTNO are recorded. At others, NULL values are recorded.

7. PRJEMP4 is created by the following JOIN:

```
JOIN TEMP WITH EMP3
INTO PRJEMP4
USING TEMP.PROJNO,TEMP.EMPNO,
```

```
EMP3.TIME,EMP3.JOBCLS
WHERE TEMP.EMPNO = EMP3.EMPNO
```

More changes are recorded in this relation in the next step.

8. NULL values should be inserted in all VAs at time points in which objects do not exist.

Assume that we have created the full range of regular relations representing the new TOR PRJEMP. Then, we can take a snapshot from this TOR at 831215, and compare it to the result of joining the corresponding snapshots of PROJECTS and EMP at the same day (Table 7-25). This snapshot is included in Table 7-27.

The Key		The CAs			The VAs		
PROJNO	EMPNO	PROJNM	NAME	SEX	COST	DEPTNO	JOBCLS
1000	10025	MDA	OSCAR	M	NULL	NULL	NULL
1000	10090	MDA	SUSAN	F	NULL	NULL	NULL
1000	10010	MDA	MIKE	M	NULL	NULL	NULL
1000	10005	MDA	MARY	F	NULL	NULL	NULL
1000	10030	MDA	HENRY	M	NULL	NULL	NULL
1000	10080	MDA	ALICE	F	NULL	NULL	NULL
1000	10050	MDA	DAVID	M	9000	1	2
1010	10010	AI	MIKE	M	NULL	NULL	NULL
1010	10005	AI	MARY	F	NULL	NULL	NULL
1010	10030	AI	HENRY	M	NULL	NULL	NULL
1010	10080	AI	ALICE	F	22000	3	2
1010	10050	AI	DAVID	M	NULL	NULL	NULL
1020	10010	TDMS	MIKE	M	18000	2	2
1020	10030	TDMS	HENRY	M	18000	2	1
1020	10080	TDMS	ALICE	F	NULL	NULL	NULL
1020	10005	TDMS	MARY	F	18000	2	3
1020	10050	TDMS	DAVID	M	NULL	NULL	NULL

Table 7-27: A Snapshot from PRJEMP at 831215, taken Directly from the TOR

This JOIN, like all other JOIN operations covered by this chapter, satisfies the strong correctness criterion. Comparing Table 7-27 with Table 7-25 leads to the conclusion that they are equivalent. The additional tuples in Table 7-27, are those for which the values of all VAs are NULL. These NULL values indicate that the objects to which they belong do not exist at 831215. The existing tuples in the two tables are, in fact, identical, hence the two tables are equivalent.

7.7. Summary

In the analysis of the JOIN operations in this chapter, we have seen that attributes may change their types in the new TOR. Table 7-28 summarize our findings in this issue.

The Type of the Common Attribute			The Types of Other Attributes in the Resulting TOR					
in the First TOR	in the Second TOR	in the Resulting TOR	The Attributes of the First TOR			The Attributes of the Second TOR		
			Key	CA	VA	Key	CA	VA
Key	Key	Key	---	CA	VA	---	CA	VA
Key	CA	CA	---	CA	VA	Key	CA	VA
Key	VA	VA	---	VA	VA	Key	CA	VA
CA	CA	CA	Key	CA	VA	Key	CA	VA
CA	VA	VA	Key	CA	VA	Key	CA	VA
VA	VA	VA	Key	CA	VA	Key	CA	VA

Table 7-28: The Types of Attributes in the Result of JOIN

In symmetric cases, all the attributes maintain their types. In the non-

symmetric cases, the common attribute inherits its type from the TOR in which it is "less stable" (according to the decreasing order of stability: Key, CA, VA). Once a common attribute inherits the VA type from an operand, all the other attributes, in the TOR in which it is not a VA, become VAs too. All these observations can be explained through the functional dependencies among all the attributes participating in a JOIN operation.

The analysis and the design of all the JOIN operations included in this chapter show that the concept of the temporal differentiation of attributes allows us to handle the most complicated operations by focusing on one attribute at a time. In addition, it allows us to create the relations representing the new TORs, one relation at a time, by executing a sequence of operations involving only the relevant relations in the operands. Using this concept for implementation design, allows the decomposition of higher level operation on TORs into a sequence of well defined operations on regular relations, creating a new sequence of relations representing the resulting TOR, thus making these operations directly executable.

Chapter 8

Summary of the Research Results

This exploratory study sheds further light on the nature of TODBs and their complexity, especially with respect to the richness of their operations. A multi-layered conceptual structure for TODBs has guided the study of an implementation model that interrelates external user views with an underlying functional view of the data, and addresses the translation of operations among these multiple layers. The actual design and prototype implementation allows us to demonstrate in full detail the properties of a relational TODB, the way it is stored and the definitions of its operations. This research has particularly emphasized the design and implementation of a general purpose TDMS that is a formal extension to the regular relational database model.

The implementation of a general purpose TODB in this dissertation is based on the evolving body of theory represented by [Clifford 82a], [Clifford 82b], [Clifford 83b], [Ariav 83a], [Ariav 84], [Ariav 85] and [Clifford 85a]. A major concept in our research has been the differentiation of the attributes according to their temporal variation. Another major concept has been the use of regular relations to implement the underlying data structures. This dissertation uses these concepts as an implementation strategy for relational TODBs, and provides a preliminary assessment of it, as applied to the three

basic aspects of DBMSs: data structures, integrity constraints and operations.

In the following sections, we point out the major findings and contribution of this dissertation in each of the major aspects of TODBs.

8.1. The Temporal Relational Algebra Operations

The issues raised in this research with respect to operations in temporally oriented DBMSs, focused mainly on a "natural" temporal extension of the regular relational algebra operations and the effect of attribute types (key, CAs, VAs) on the algorithmic definitions of these operations.

The discussions in chapters 4, 5, 6 and 7 dealt with issues. We have defined in general terms the extensions to the regular relational algebra operations, and then had specific discussions covering the temporal SELECT, PROJECT and JOIN operations. The temporal PROJECT and JOIN operations are strictly natural extensions to the corresponding regular operations, in the sense that their various time slices are equivalent to the results of the same operations on the corresponding time slices of the operand(s). The temporal SELECT operation has two versions, the SELECT SOMEWHEN and the SELECT EVERYWHEN. Both SELECT operations can only conform to the weak correctness criterion. All the temporal operations maintain the closure property of the algebra, creating new TORs as their results. In addition, two other temporal operations were defined, the time selection and the time projection, which are unique temporal operations. The major properties of the temporal operations are discussed in the following subsections, analyzing each of them separately.

8.1.1. The SELECT Operations

Three SELECT operations were defined and implemented in the TDMS: the SELECT SOMEWHEN, the SELECT EVERYWHEN and the time selection.

The SELECT SOMEWHEN, as defined in Chapter 5, selects all the data of every object that has at least one qualifying tuple, and records it in the resulting TOR. Therefore, it selects tuples that might not qualify in themselves, but belong to a qualifying object. This operation cannot be properly defined in terms of the various time slices comprising the cube, since conceptually it does not "operate" on one time slice at a time, but on one object at a time. If the entire horizontal slice of each object contains at least one qualifying tuple, then the entire history of that object qualifies.

The SELECT SOMEWHEN operation satisfies only the weak correctness criterion as defined in Chapter 4. It does not satisfy the strong criterion for conceptual reasons, namely the slightly different interpretation of this operation compared to the regular SELECT. Had we adopted an alternative definition, according to which only qualified tuples would have been selected, this operation would have satisfied the strong criterion, but would have had serious deficiencies, expressed by recording wrong information for non-qualifying tuples of qualifying objects (see [Ariav 83a]).

The SELECT EVERYWHEN operation selects only those objects that satisfy the selection predicate at all time points. If the operand contains only one time slice (which is informationally equivalent to a regular relation), then

the SELECT SOMEWHEN and the SELECT EVERYWHEN have the same meaning. Therefore, the two operations can be considered as extensions to the regular SELECT operation.

The SELECT EVERYWHEN operation satisfies only the weak correctness criterion. As mentioned before, it does not satisfy the strong criterion since selecting tuples on a "one time-slice at a time" basis, introduces tuples of objects that do not satisfy the selection criterion EVERYWHEN.

A unique temporal operation, the time selection, was also defined in the TDMS. This operation creates a new TOR containing the data of a specified time interval. Since it is not an extension to a regular operation, applying any of the correctness criteria to this operation is irrelevant.

8.1.2. The Temporal PROJECT Operation

The temporal PROJECT operation is the temporal extension to the regular PROJECT. Every time slice of the resulting TOR is the result of the same PROJECT on the corresponding time slice of the operand. This operation includes two cases: the PROJECT operations that preserve the key, and those that do not. The PROJECT operations that preserve the key create new TORs with the same objects as in the operands, but with fewer attributes. Consequently, all the operands' objects are included in the resulting TOR.

In a PROJECT operation that does not preserve the key the original objects lose their identities, new objects are created, and the new TOR has therefore a new meaning depending on the projected attributes. This situation

is not caused by the temporal nature of TODBs; the same happens in regular PROJECT operations. However, in the temporal case, not only the meaning of such operations is different from those that preserve the key, but their implementations are more complicated as well. The most complicated implementation procedure is the one for the case in which only non-key CAs are projected; as these attributes do not carry any temporal information, all the VAs relations are accessed in order to establish the periods of time in which the new objects exist, as inferred from the existence of objects in the operand.

If a PROJECT operation, that does not preserve the key, projects at least one VA, then this VA carries the temporal information needed to determine the periods of time in which the new objects exist; therefore there is no need to use non-projected VAs for this purpose. The implementation of such an operation is simpler than the one in which no VA is projected. However, it is more complicated as more VAs are projected, since the combination of them determines the periods of time in which the new objects exist.

In every PROJECT operation that does not preserve the key, the new key consists of all the projected attributes, since the new objects are their various combinations. Therefore, there are no VAs by which we can represent periods of time in which objects do not exist. The solution to this problem is in the introduction of a special VA into the resulting TOR. This synthetic attribute, called STATE, is aimed to allow the proper recording of the fact that objects of the new TOR may not exist at all time points. STATE has NULL values for an object at all time points in which this object does not exist, and the value 1 otherwise.

The problem that caused the introduction of the new VA STATE is inherent to the temporal nature of the TOR, and is not caused by our specific design. It can be expected that such a problem will surface under any other implementation strategy of TODBs in the case of the PROJECT operation that does not preserve the key.

The temporal PROJECT is critically affected by the types of the projected attributes. The conceptual problems in this operation coincide with the implementational problems. If the whole key is preserved, then conceptually it is a simple operation that just drops some attributes from each object. In such a case, the implementation is also very simple. The relations representing projected attributes are copied, thus creating the resulting TOR. On the other hand, the implementation of the temporal PROJECT that does not preserve the key requires more operations, as new objects are created, and their existence periods should be determined, as well as the periods of time in which they do not exist.

The PROJECT operation satisfies the strong correctness criterion, independently of whether it preserves the key or not.

8.1.3. The Time Projection Operation

The time projection does not analyze the data stored in the TOR through its time slices, but considers the whole temporal pattern of the original objects with respect to the projected attributes. Two objects in the original TOR are "merged" to the same object in the resulting TOR, only if the combinations of their projected attributes exhibit the same temporal patterns. Otherwise, they maintain their identities in the new TOR too.

As in the PROJECT operation that does not preserve the key, a new attribute, called IDENT, is added to the resulting TOR. Nevertheless, this new attribute is the new TOR's key, an attribute by which the new objects are identified.

It should be emphasized here that, even though the PROJECT operation that does not preserve the key, and the time projection operation raised unique problems that were handled by adding a new attribute to their resulting TORs, they still maintain the closure property of the temporal operations. TORs resulting from these operations could be further manipulated by any temporal operation.

According to its definition, the time projection operation does not satisfy the strong correctness criterion. The reason for this is in its conceptual definition, as it defines different objects according to their behavior along the time dimension, and not on the basis of the various time slices. Therefore, each time slice of the resulting TOR is not the result of this operation on the corresponding time slice of the operand. However, if this operation is applied to a single time slice, then it executes like a regular PROJECT operation, and therefore it satisfies the weak correctness criterion.

The implementation of the time projection depends on the types of the projected attributes. It is fairly simple; the different patterns of the combinations of the projected attributes has to be identified and copied to the resulting TOR.

8.1.4. The Temporal JOIN Operation

The temporal JOIN operation has not been satisfactorily studied in the past. Its purpose is the same as the one of the regular JOIN. Nevertheless, the difficulties involved in implementing it are substantial, since two cubes are merged here, rather than two flat tables. The JOIN operation is viewed conceptually as the union of the regular JOIN with pairs of snapshots from the operands, creating thereby new objects based on the objects in the operands. Even though there is only one conceptual version of the JOIN, in terms of the implementation, this operation is divided into six cases based on the types of the common attribute of the two operands. Each case defines a class of JOIN operations with a unique sequence of implementation activities.

A TOR, created by a JOIN operation, contains the union of attributes of the two operands. Most of these attributes are stored separately in the various regular relations representing the operands of the JOIN, and they should be stored in the new TOR in the same way. Therefore, each relation of the new TOR is typically created by manipulating two of the representing relations of the operands. The implementation of the temporal JOIN operation is accomplished by practically decomposing this operation into a sequence of manipulations with the relations representing the operands. Most of these manipulations are regular JOIN operations. In addition, a final step is required, to correctly determine the periods of time in which each object exists in the resulting TOR, based on the information of the objects that created it in the operands. This step results in recording NULL values for all the VAs of objects that do not exist in the new TOR at some time periods, as implied by the information of their original objects in the operands.

The temporal JOIN operation was successfully defined to satisfy the strong correctness criterion.

8.1.5. Summary of the Temporal Operations

We consider the definitions and the implementations of the temporal operations as the main contribution of this research. An important aspect is the distinction between the two criteria for the correctness of a temporal operation - a strong criterion and a weak one - and the examination of each operation with respect to these criteria. All the SELECT operations and the time projection operation satisfy only the weak criterion, as they are defined for the entire data of each object, and not on the basis of the operand's time slices. All other temporal operations satisfy the strong correctness criterion.

The closure property is fully maintained in the temporal operations defined in this dissertation. Any TOR, resulting from a temporal operation, can be an operand of another operation, without any restriction. Even the temporal operations that introduce new attributes into their results, namely the PROJECT operation that do not preserve the key and the time projection, create perfectly valid TORs, that can further participate in any other temporal operation.

8.2. The Temporal Differentiation of Attributes

All the aspects of TODBs covered by this research exploit the temporal differentiation of attribute, a concept which has proven to be beneficial for implementing TODBs and TDMSs. This section summarize our findings in this issue.

8.2.1. Temporal Differentiation of Attributes and Temporal Operations

The definition of the temporal relational algebra operations at the implementation level is the main benefactor of the temporal differentiation of the attributes. This concept, as applied to the temporal operations, allows us to delineate easily the scope of each operation, by concentrating on the relation(s) containing the data items affected by these operations. In some cases, only a few new relations have to be generated to represent the new TORs created by the operations, while some of these relations are simply identical to existing relations that underly the operand(s). This concept allowed us to decompose each operation to a sequence of manageable operations with regular relations.

In the two versions of the SELECT operations, the temporal differentiation of the attributes helps in identifying the qualifying objects by accessing only those relations that represent attributes that are included in the predicate. The entire information of these objects is then copied to the new TOR, and therefore all the relations representing the operand should be accessed. The resulting TOR contains a subset of the operand's objects, and as no new objects are created, the implementation of these operations is

achieved by relatively simple manipulations of the relations representing the operand.

The implementation of the time selection operation does not benefit from the temporal differentiation of attributes as all the relations representing the operand have to be accessed when the relevant data is copied from the original relations to the new ones.

The implementation of the PROJECT operation that preserves the key takes full advantage of the temporal differentiation of attributes, since most of the new relations are simply copied from the relations representing the operand. However, in implementing the PROJECT operation that does not preserve the key, the temporal differentiation of attributes is not advantageous, and may even complicate the implementation of this operation, by causing a simultaneous scanning of the relations representing the VAs involved. Such a scanning may not appear under different implementation strategy. In this operation, we also have the problem of introducing the artificial VA STATE. This problem, however, is not caused by the temporal differentiation of attributes, but by the fact that new objects are defined by the combination of all the projected attributes.

The implementation of the temporal JOIN takes full advantage of the temporal differentiation of attributes. Our approach allows us to decompose the global problem of designing the JOIN, to a sequence of manageable operations, each of which aims to build one new relation as a result of the necessary manipulation with the designated relations of the operands.

8.2.2. Temporal Differentiation of Attributes and Integrity Constraints

Integrity issues in our research centered on integrity checks and their execution in TDMS. Under the temporal differentiation of attributes, each logical unit (an attribute) is also a physical unit, and hence checking a logical unit is rather simple, and typically only little irrelevant data is accessed.

While adding a new object to a TOR, the TDMS verifies that such an object does not yet exist. To do this, only the relation containing the CAs of this TOR has to be accessed and checked. Whenever an object is added to a TOR, values (other than NULL) should be assigned to all its CAs. From this point on, no changes can be made to the CAs of this object. After adding a new object, there is no need to assign any values to any of its VAs. They are automatically interpreted as NULL, as long as no other values are assigned to them.

The temporal differentiation of attributes plays a major role in the procedure to append data pertaining to existing objects. Verifying the existence of an object requires accessing only the CAs relation, and the maintenance of the finality property requires accessing only the relation containing the data of the VA involved. Therefore, consistency is maintained efficiently, and the involvement of relations in the integrity checks is limited to those directly affected by the loading operation.

8.3. NULL Values and Object Existence

The value of a time varying attribute may happen to be unknown at some time point, and therefore this VA may assume a NULL value (see Chapter 3). In this dissertation only one type of NULL was specified and used. This can later be extended to contain different NULL values for different situations in which the attribute value is unspecified (see [Clifford 85a]).

In order to represent time points at which an object does not exist, we use the convention of assigning NULL values to all the VAs of this object at such time points. A different convention could be to add a "hidden" system attribute to each TOR, which indicates directly such situations. It should be noted, however, that NULL values would have been recorded anyway in the VAs of non existing objects during periods of their inexistence. Otherwise, the final determination of the value of a VA at any time would always require simultaneous checking of the VA that contains existence information for objects. Another alternative to indicate inexistence intervals could be the recording of time spans for each VA in the TOR. As any approach cannot avoid dealing with the problem of indicating somehow the time points in which an object does not exist, our convention seemed a legitimate choice. It does not require any additional facilities, but takes advantage of the existing components of the TOR, to indicate non-existence situations.

The role of NULL values in our TDMS requires special attention in executing the temporal operations. The various algorithms should record NULL values in the VAs relations of the resulting TOR, to indicate possible non-

existence of its objects, as implied by non-existence of objects in the operand(s). The problem of indicating non-existence periods of time of objects in a TOR was expected, since it is a direct result of the external cubic view of the TOR. Our solution to this problem did not complicate the design of the data structures, and allowed us consistent treatment of all the TDMS's components. We feel that our means of handling these situations is a self-contained one that allows us to maintain consistency in all the operations executed with TORs. However, since we experienced problems with our design (e.g., in resolving a PROJECT operation that does not preserve the key), and since other methods can be suggested, we recommend, in the conclusion of this dissertation, to conduct more research concerning the use of NULL values in general, and their particular role as indicating non-existence of objects at specific periods of time.

8.4. Implementation Aspects

The use of regular relations as the primitive objects in our implementation-level model, grants the following advantages:

* Using an Existsing DBMS

The representing relations are manipulated by an existing relational DBMS, which implies a significantly reduced effort in building a working TDMS prototype. This approach has probably resulted in an inferior run-time performance. However, as we are still exploring the fundamental conceptual properties of these systems, we are not concerned with performance.

The feasibility of using an existing relational DBMS was demonstrated by

our design. It allowed us to build the prototype on top of INGRES within a limited period of time, and save the effort involved in building the I/O modules in the TDMS.

* Preventing the need to rearrange data

Each data item appended to the database, is actually appended to a specific relation. Whenever appending a data item to a TOR, there is no need to rearrange old data; the new data item will be correctly interpreted whenever used.

* Implementing the Temporal Operations

In implementing the temporal operations, some problems related to our specific implementation approach were detected. In the following paragraphs we briefly comment on them.

In implementing the temporal JOIN, the types of the attributes in the resulting TOR are determined, based on the types of the common attribute in the two operands, and on the types of each attribute being incorporated into the resulting TOR. The situation according to which attributes change their types in a JOIN operation occurs in the regular case as well (e.g., key attribute may become non-key attribute). In the TDMS, it affects the implementation, since the attributes' types play a major role in determining their storage.

An unexpected difficulty in designing the TDMS involved the execution of PROJECT operations that do not preserve the key. In executing these

operations, many relations have to be simultaneously scanned, in order to identify the non-existence time periods of the new objects. The implementation of the temporal JOIN is complicated too, but this is no surprise since the operation is inherently complicated. We do not think that any other implementation strategy, or other techniques (such as indicating non-existing periods of time of objects) can avoid the difficulties involved in implementing the temporal JOIN.

* Comments on the Design of TDMS

In the following paragraphs, we highlight the decisions that were made during the process of designing the TDMS, and point out their ramifications.

The first decision was that TOR's key will include constant attributes (CAs) only. This allowed us to store all the keys of the TOR's objects in one compact relation (the CAs relation), and then link through this key objects with their temporal data in the VAs relation. If at some point the key of some object does change, the TDMS can handle that by "closing" the existing object, and creating a new object with values inherited from the old one. The values in the time slice of this time point should be copied from the old object to the new one, and then NULL values should be recorded for the old object in all its VAs for this time point, marking it as non-existent. Having limited the key to CAs, helps in implementing the temporal operations, as object identification and matching are relatively simple.

Another point that should be emphasized is that a TOR does not have to have either CAs or VAs. The only component that should exist in the TOR is the

key. Therefore, if there is a doubt whether an attribute may be subject to change, it could be defined as a VA and not as a CA. This may cost some extra space, since this attribute will be stored in a separate relation, rather than in the CAs relation, but it grants the required flexibility in handling this attribute.

8.5. Performance Evaluation

This section examines some performance aspects of the TDMS, namely its storage requirements and the complexity of its operations.

* The Data Structures

With respect to the TDMS's data structures, we were able to represent TORs by a sequence of regular relations which contain the minimal amount of information needed to construct the whole cube. Each time varying attribute is represented by a separate relation, all the constant attributes (including the key) of a TOR are stored together in a single relation and one additional relation contains the TOR's schema. In the representing relations of a TOR, the key of the TOR has to be recorded in every VA relation. To quantify this redundancy, assume that the key of the TOR contains k attributes, the TOR contains v VAs, and the maximal number of tuples in any VA relation is t. Then, the order of storage redundancy is O(k*t*v).

* Storage Space

In terms of storage space, our data structures appear to be generally more efficient than the methods used in related research efforts e.g., [Snodgrass 84] and [Lum 84]. In those works, the time stamps are associated

with the whole tuple. Therefore, if the value of one attribute in some tuple changes, the entire tuple has to be re-recorded with the new time stamp, even though only one value has changed. Since the time stamps in our design are associated with the various attributes, and not with tuples, there is no redundancy in storing the data, and whenever a new value should be associated with some attribute of a specific object, there is no need to duplicate the old values of this object. Under the assumption that most the VAs in a TOR do not change simultaneously, our method would require less memory space.

* The Operations Complexity

The representation of a TOR by a sequence of regular relations allows also for efficient algorithms to execute the temporal operations. In the following paragraphs we summarize this issue.

The algorithms to execute any of the SELECT operations (the SELECT SOMEWHEN, the SELECT EVERYWHEN and the time selection operation) are fairly simple and not very costly in terms of performance. If the operand contains n objects, and the maximal number of tuples per object in all the representing relations is m, then the number of operations needed to execute any of these operations is O(n*m).

The execution of a PROJECT operation that preserves the key carries the maximal benefit from our data structures. if the maximal number of tuples in a representing relation of the operand is t, then the number of operations in our implementation of this PROJECT is O(t). The number of operations needed to execute a PROJECT operation that does not preserve the key is substantially

higher. If the operand contains n objects, then the resulting TOR has at most n objects. For each of them, some of the VAs relations (depending on the specific PROJECT) have to be simultaneously scanned. If the maximal number of tuples per a VA relation is u, then the whole operation requires O(n*u) database operations. The same is true for the time projection operation.

The implementation of the JOIN operation requires two major steps. In the first step the new representing relations are created without determining the time points in which the new objects do not exist. In this step, each new relation is created by either a regular JOIN with two relations representing the operands, or by another operation with such relations, that is equivalent to a regular JOIN in performance terms. Therefore, if the maximal number of tuples in any of the VA relations of the first operand is u_1, and that of the second operand is u_2, then this step requires $O(u_1*u_2)$ database operations. Then, in the next step, each object of the resulting TOR is checked against the objects of the operands that created it, to record the proper NULL values, whenever needed. If the number of objects in the first operand is n_1, and in the second operand n_2, and the maximal number of tuples per object in any of the VAs relations of the operands is m, then this operation requires $O(n_1*n_2*m)$ database activities. This operation is very costly in terms of performance. It should be noted, however, that it can be "smartly" executed, to save unnecessary computer time. Typically, in a JOIN operation the user is not interested in having all the original attributes in the resulting TOR, but only those related to this operation. Therefore, before executing a temporal JOIN, it is recommended to PROJECT the two operands onto the relevant

attributes. This PROJECT operation preserves the key, therefore is very simple, takes the maximal advantage of the temporal differentiation of attributes, and is efficient in terms of performance. The new operands contain fewer attributes and may lead to a more efficient JOIN.

8.6. Summary

This chapter briefly summarized the major findings of the research. We feel that even though our research could not examine all the alternatives concerning the TDMS design, it has created enough information to point out possible directions towards the developing of such systems.

Chapter 9

Conclusion and Future Research

This chapter points out the major contribution of this research (Section 9.1), outlines some immediate potential improvement to our TDMS prototype (Section 9.2) and raises topics requiring more research in the areas of relational TODBs (Section 9.3).

9.1. The Contribution of this Research

This research focuses on the design and implementation of relational Temporally-Oriented Database Management Systems (TDMSs). Our design presents an efficient and flexible structure for a TDMS. This has been achieved by two major properties of our implementation model: the temporal differentiation of attributes and the use of regular relations as primitive objects by which the Temporally Oriented Relations (TORs) are represented. The power granted to our TDMS by these properties is demonstrated many times throughout the research.

In building our prototype, we have used the interpretive approach, and the manipulations of the representing relations are carried out by the existing DBMS INGRES. By so doing, we have demonstrated the ability of using an existing DBMS, rather than starting from scratch. This implementation strategy relieved us from designing the basic I/O mechanisms, and allowed us to concentrate on the application of the TDMS itself. There are, of course,

trade-offs between this approach and the other alternative, namely, developing the entire data management mechanism from scratch, which would probably produce a better performing system. However, for an exploratory research like ours, the advantages of the interpretive approach, allowing us to build the prototype in a much shorter period, are clear, and justify our choice.

The functional TDMS prototype is beneficial to future research in this area. This prototype can be used for further research in aspects not covered by this dissertation, such as query optimization and concurrency control.

This implementation research has been the first to spell out specifically a complete set of temporal relational algebra operations. Furthermore, the design of the temporal JOIN operation is a significant contribution, since this operation has not been defined in previous research (as opposed to the SELECT and PROJECT operations), and its implementation in our research has been feasible mainly due to the use of the temporal differentiation of attributes.

9.2. System Extensions and Improvements

Our TDMS is doubtlessly not complete. The major immediate potential extensions and improvements that can be introduced to it are:

* Improving the TDMS's performance

The TORs in our TODB are represented by regular relations. Therefore, the implementation of the temporal operations basically creates a new sequence of relations to represent the resulting TOR, by manipulating the relations

representing the operand(s). In many situations, some of the original relations are copied to the new ones without any change. We consider this to be one of the advantages of the temporal differentiation of attributes, which allows us to concentrate on the relations directly involved in the operation, and ignore or just copy other relations.

However, we could just place the names of the old relations in the descriptive relation of the new TOR (without, of course, deleting them from the descriptive relations of the original TOR[s]), and thereby have the same relations participating simultaneously in more than one TOR. In so doing, we sometimes could create a new TOR as a result of a temporal operation with extremely few operations. We virtually have just to create a new descriptive relation, and possibly a small number of new relations. This approach could be especially useful in PROJECT operations that preserve the key.

The second approach would definitely be more efficient in terms of memory space and possibly also in execution time. However, more research is needed to examine all its implications. For instance, it could only be possible to actually delete a TOR if no other TOR "pointed" to it,

* Generalization of the system's operations

Basically, both temporal relational algebra operations and temporal operations have been defined, implemented and demonstrated in this research. However, we cannot claim that they form a complete set of operations. They have some weak points that require more research. The following is a list, not necessarily complete, of some weaknesses already detected by us, concerning the TDMS's operations:

1. So far, only constants are allowed in the predicates of the temporal relational algebra operations. An immediate improvement that should be applied to the TDMS is the option to include variables in these predicates.

2. Chapter 6 analyzes the problem of PROJECT operations that do not preserve the key, and proposes a solution. More research is needed to examine other possible solutions that may be better than ours.

3. The implementation of the JOIN operation is limited to the cases in which only one attribute is common to the operands. Having more than one common attribute may definitely increase the complexity of the problem, and more research is needed in this issue too.

4. Our TDMS uses only one type of NULL values. Previous research efforts, such as [Ariav 83a] and [Clifford 85a], have already proposed more than one NULL value in TODBs. Their effect on the temporal operations should be investigated.

5. Existing relational DBMSs typically provide not only relational algebra operations, but also reductive operations such as calculating the sum or the average of attributes, count them, etc. Such operations should be incorporated into the TDMS too.

6. Most of the temporal operations in our TDMS are natural extensions to the regular relational algebra operations. Those operations are conceptually defined on the basis of one time slice at a time. There are no operations in which two different time slices are manipulated in the same conceptual step. Let us elaborate on this point with respect to the SELECT SOMEWHEN and to the JOIN operations. In a SELECT SOMEWHEN operation, a query may select all employees whose salaries at some time point is lower than their salaries one month earlier. A more complicated query could select employees whose salaries were decreased at some time points. In a JOIN operation, one could think about a JOIN operation that conceptually joins each time slice of the first operand with the time slice corresponding to some lagged time point in the second operand. These two examples present another class of temporal operations, that, as opposed to the operations included in this research, are not equi-temporal operations. The implementation of the temporal operations, as included in this dissertation, is the first step in designing non-equi-temporal operations, and more effort should be made to augment the existing operations in this direction.

* Query languages

The language used in our system is strictly a procedural one. There is a lot to be done to incorporate non-procedural query languages like those suggested by [Snodgrass 84] and [Gadia 84]. The issue of incorporating a capability in the TDMS to analyze a relational calculus expression, and decide what operations should be executed to create its result, requires extensive research. In particular, an investigation should be conducted to explore the methods used in the regular relational DBMSs that can be extended to be used in TDMSs. More research is also needed to extend the notion of a relationally complete query language, and apply it to the temporal model.

* Grouping of attributes

The present design assumes that all the VAs in a TOR have different temporal variations. However, there may be special applications in which there are some attributes that by their nature have the same temporal variation, For example, the TOR describing employees may have two VAs, RANK and SALARY, that may change simultaneously. Such attributes can be handled in a more efficient way by being grouped to one data structure that will maintain one time stamp for all of them, rather than having the same time stamps associated with each of them separately. This idea suggests a possible extension of our model, that will be achieved by grouping such attributes to be included in one relation, with one time stamp. This grouping will presumably save space and processing time, in the price of giving up the ability to handle any of these attributes separately.

9.3. More Future Research Topics

* Evolving Schemes

From a broader perspective of TODBs, not only historical data is to be maintained, but also historical definitions of the TORs in it [Ariav 83a]. This raises the issue of database restructuring as dealt with by [Sockut 79] and [Navathe 80]. This capability can accommodate situations in which new types of data become available at some point, but the database cannot be changed retroactively. In these cases, the schemes of the relevant TORs are modified accordingly, but the definitions for the older data still underly the execution of operations on old versions of these TORs ("old" in the sense that they do not contain the new data item). A basic research is needed to identify the issues involved in implementing the evolving schemes capability into TDMSs in general, and to incorporate it into our model in particular.

* Performance Evaluation

This research does not contain a performance evaluation of the TDMS prototype. The common expectation is that improved treatment of temporal aspects, as represented by systems like TDMS, will imply lower DBMS performance, in terms of CPU and I/O operations, storage space, response time, etc. In general, performance evaluation can be pursued through theoretical analysis and/or practical methods.

Following, is a brief analysis of possible approaches to performance evaluation research. Once there are several designs (and possibly prototypes) of TDMS, developed under different implementation strategies, there will be an opportunity to conduct research comparing their performance.

The various algorithms that are used to perform the operations on TORs can be analyzed to determine the relevant performance factors involved, such as number of accesses to regular relations, the number of tuples brought into memory, the analytical complexity, etc., as functions of the properties of the TORs being manipulated, such as the number of their attributes and their types, the sizes of the relations representing them, etc. These results can then be compared to similar measures, already known, associated with the various relational algebra operations on regular relations. This comparison may give us some appreciation of the additional computations executed when using a TDMS. However, it is almost sure that it is difficult to define a meaningful comparison, since we have two different types of databases (TODB vs static database), and operations are performed differently within them.

A different approach to performance evaluation could be the actual measurements of benchmark runs. According to this technique, actual measurements are taken through monitoring the run-time behavior of experimental database(s), under a representative sample of operations. These runs can provide us with two types of data namely, OS-provided overall performance measurements (e.g., total consumption of CPU time, number of I/O operations, disk-storage occupation) and a set of measurements that can be collected by our system itself, to reflect major operational factors in high level terms (number of accesses to regular relations, amount of data moved to main memory, number of internal operations). Such measurements can then be compared to similar measurements collected during a process that produces the same results under different conditions and different TDMSs.

* Query Optimization

Research may discover ways to take advantage of the unique nature of TODBs for the sake of query optimization. We expect to find that the independent existence of the attributes has a major impact on this subject too.

* Concurrency Control

Basic research is needed in this area to shed light on the specific problems of concurrency control (if any) in TODBs, and the ways to handle them. Again, the temporal differentiation of attributes may be beneficial with this respect as well. Whenever data is being loaded into the database, there is no need to lock the whole TOR involved, but only the attribute(s) to which new values are added, since each attribute is stored in a separate physical unit.

* Handling More than One Temporal Dimension

Our system handles one temporal dimension, e.g., the physical time. If more temporal dimensions are needed, e.g., the recording time (to deal with AS-OF time), then they have to be stored explicitly by the user. The inclusion of more than one temporal dimension in a working system presents fundamental research problems that have to be further investigated.

* Using Temporal Spans for Attributes

In our model, each VA varies along the time dimension without any formal limits. Its values before the first explicitly recorded value are determined

by interpolation (typically NULL values). The values after the last recorded value are determined by this interpolation as well (typically, the last recorded value). A different approach could suggest that each VA in each TOR will have two time points associated with it, indicating the first time point and the last time point at which this attribute's value is valid. Conceptually, this modification converts the cubic view of the TOR to an image constructed from several cubes that are pasted together. In practice, the implications of such a modification on all the aspects of the TDMS (data structures, operations, constraints, etc.) should be a subject of further research.

* Updating Views

The append operation was restricted to base TORs, created originally by the user, and not to views, created by the temporal relational algebra operations. This restriction seems to be needed to prevent the introduction of inconsistencies into a TODB. This is again a subject for future research, that can identify ways to allow the updating of views under a set of appropriate rules.

* Indication of Non-Existing Tuples

In our TDMS, the indication of a non-existing tuple of an object in a TOR at a specific time point, is the assignment of NULL values to all its VAs at this time point. This is not the only way to indicate such situations. Another way could be the addition of an invisible attribute to every TOR, whose only purpose is to indicate the time points at which objects exist, and

the time points at which they do not exist. The implications of the various alternative methods to indicate the existence of objects at the various time points is another issue that requires future research.

9.4. Closing Remarks

This study is one further step in the process of making TDMS, the temporal extension of the regular DBMS, as common and useful as the existing DBMSs. This research has not covered all the aspects of implementing TODBs completely and comprehensively. Nevertheless, it has created a sharper and clearer picture of the complexities inherent in this topic, which should guide us in identifying further research issues in this rich domain.

Appendix and References

Appendix A

The Content of the Benchmark Database

The benchmark database included in this dissertation contains nine TORs, and serves to demonstrate the TDMS's capabilities throughout the dissertation. The contents of these TORs, in terms of their internal views and their representing relations, are presented in the following tables.

EMP(EMPNO,NAME,SEX,DEPTNO,JOBCLS)

EMPNO	NAME	SEX	DEPTNO	JOBCLS
10010	MIKE	M	800101 3	800101 4
			810215 2	810201 3
				821015 2
10005	MARY	F	810210 2	810210 3
10050	DAVID	M	800601 1	800601 3
			820508 NULL	820508 NULL
			830415 1	830415 2
10030	HENRY	M	800101 2	800101 3
			820701 3	820101 2
			830508 2	830304 1
10080	ALICE	F	810101 3	810101 2
10025	OSCAR	M	800101 4	800101 1
10090	SUSAN	F	800101 4	800101 4
				811015 3

Table A-1: The Internal View of the TOR EMP

EMP(ATTRIBUTE,PTYPE,LTYPE)

ATTRIBUTE	PTYPE	LTYPE
EMPNO	I4	1
NAME	C20	2
SEX	C1	2
DEPTNO	I2	3
JOBCLS	I2	3

EMP1(EMPNO,NAME,SEX)

EMPNO	NAME	SEX
10010	MIKE	M
10005	MARY	F
10050	DAVID	M
10030	HENRY	M
10080	ALICE	F
10025	OSCAR	M
10090	SUSAN	F

EMP2(EMPNO,TIME,DEPTNO)

EMPNO	TIME	DEPTNO
10010	800101	3
10010	810215	2
10005	800101	2
10050	800601	1
10050	820508	NULL
10050	830415	1
10030	800101	2
10030	820701	3
10030	830508	2
10080	810101	3
10025	800101	4
10090	800101	4

EMP3(EMPNO,TIME,JOBCLS)

EMPNO	TIME	JOBCLS
10010	800101	4
10010	810201	3
10010	821015	2
10005	810210	3
10050	800601	3
10050	820508	NULL
10050	830415	2
10030	800101	3
10030	820101	2
10030	830304	1
10080	810101	2
10025	800101	1
10090	800101	4
10090	811015	3

Table A-2: The Regular Relations Representing the TOR EMP

SAL(EMPNO,SALARY)

EMPNO	SALARY
10025	800101 30000
	811120 32300
10010	800101 19500
	820215 22100
10005	810210 20300
	830101 22500
10050	800601 21200
	820508 NULL
	830415 23500
10030	800101 22000
	810601 23500
	831015 24500
10080	810101 24000
10090	800101 19700
	820101 21200

Table A-3: The Internal View of the TOR SAL

SAL(ATTRIBUTE,PTYPE,LTYPE)

ATTRIBUTE	PTYPE	LTYPE
EMPNO	I4	1
SALARY	F4	3

SAL1

EMPNO
10025
10010
10005
10050
10030
10080
10090

SAL2(EMPNO,TIME,SALARY)

EMPNO	TIME	SALARY
10025	800101	30000
10025	811120	32300
10010	800101	19500
10010	820215	22100
10005	810210	20300
10005	830101	22500
10050	800601	21200
10050	820508	NULL
10050	830415	23500
10030	800101	22000
10030	810601	23500
10030	831015	24500
10080	810101	24000
10090	800101	19700
10090	820101	21200

Table A-4: The Regular Relations Representing the TOR SAL

DEPT(DEPTNO,DEPTNM,DEPMGR)

DEPTNO	DEPTNM	DEPMGR
1	SALES	800601 10050
		820508 NULL
		830415 10050
2	PRODUCTION	800101 10030
		820701 10005
3	ACCOUNTING	800101 10010
		810215 10080
4	MANAGEMENT	800101 10025

Table A-5: The Internal View of the TOR DEPT

DEPT(ATTRIBUTE,PTYPE,LTYPE)

ATTRIBUTE	PTYPE	LTYPE
DEPTNO	I4	1
DEPTNM	C12	2
DEPMGR	I4	3

DEPT1(DEPTNO,DEPTNM)

DEPTNO	DEPTNM
1	SALES
2	PRODUCTION
3	ACCOUNTING
4	MANAGEMENT

DEPT2(DEPTNO,TIME,DEPMGR)

DEPTNO	TIME	DEPMGR
1	800601	10050
1	820508	NULL
1	830415	10050
2	800101	10030
2	820701	10005
3	800101	10010
3	810215	10080
4	800101	10025

Table A-6: The Regular Relations Representing the TOR DEPT

COURSE(CRSNO,CNAME,PRICE,DURATN)

CRSNO	CNAME	PRICE	DURATN
100	BASIC	760101 100	760101 5
		820107 120	800710 6
200	FORTRAN	791001 0	791001 3
		810215 50	820101 5
		821020 70	830215 10
		830501 130	
150	COBOL	770515 150	770515 8
		810210 200	801201 10
			820112 12

Table A-7: The Internal View of the TOR COURSE

COURSE(ATTRIBUTE,PTYPE,LTYPE)

ATTRIBUTE	PTYPE	LTYPE
CRSNO	I2	1
CNAME	C20	2
PRICE	F4	3
DURATN	I2	3

COURSE1(CRSNO,CNAME)

CRSNO	DEPTNM
100	BASIC
200	FORTRAN
150	COBOL

COURSE2(CRSNO,TIME,PRICE)

CRSNO	TIME	PRICE
100	760101	100
100	820107	120
200	791001	0
200	810215	50
200	821020	70
200	830501	130
150	770515	150
150	810210	200

COURSE3(CRSNO,TIME,DURATN)

CRSNO	TIME	DURATN
100	760101	5
100	810201	6
200	791001	3
200	820101	5
200	830215	10
150	770515	8
150	801201	10
150	820112	12

Table A-8: The Regular Relations Representing the TOR COURSE

TRNHST(EMPNO,CRSNO,GRADE)

EMPNO	CRSNO	GRADE
10010	100	800815 80
		820310 92
10010	200	810515 78
10050	150	800910 85
10050	200	810515 70
		831210 90
10050	100	830815 88

Table A-9: The Internal View of the TOR TRNHST

TRNHST(EMPNO,CRSNO,GRADE)

ATTRIBUTE	PTYPE	LTYPE
EMPNO	I4	1
CRSNO	I2	1
GRADE	I2	3

TRNHST1(EMPNO,CRSNO)

EMPNO	CRSNO
10010	100
10010	200
10050	150
10050	200
10050	100

TRNHST2(EMPNO,CRSNO,TIME,GRADE)

EMPNO	CRSNO	TIME	GRADE
10010	100	800815	80
10010	100	820310	92
10010	200	810515	78
10050	150	800910	85
10050	200	810515	70
10050	200	831210	90
10050	100	830815	88

Table A-10: The Regular Relations Representing the TOR TRNHST

DRESS(SEX,ROOM)

SEX	ROOM
M	800101 M304
	820512 M404
F	800101 M610

Table A-11: The Internal View of the TOR DRESS

DRESS(ATTRIBUTE,PTYPE,LTYPE)

ATTRIBUTE	PTYPE	LTYPE
SEX	C1	1
ROOM	C4	3

DRESS1(SEX)

SEX
M
F

DRESS2(SEX,TIME,ROOM)

SEX	TIME	ROOM
M	800101	M304
M	820512	M404
F	800101	M610

Table A-12: The Regular Relations Representing the TOR DRESS

UNIONS(UNION,SEX,OFFICE)

UNION	SEX	OFFICE	
ALPHA	M	800501	M101
		810615	N503
BETA	F	800601	M102
		810620	N505
GAMA	M	810210	W203

Table A-13: The Internal View of the TOR UNIONS

UNIONS(ATTRIBUTE,PTYPE,LTYPE)

ATTRIBUTE	PTYPE	LTYPE
UNION	C8	1
SEX	C1	2
OFFICE	C4	3

UNIONS1(UNION,SEX)

UNION	SEX
ALPHA	M
BETA	F
GAMA	M

UNIONS2(UNION,TIME,OFFICE)

UNION	TIME	OFFICE
ALPHA	800501	M101
ALPHA	810615	N503
BETA	800601	M102
BETA	810620	N505
GAMA	810210	W203

Table A-14: The Regular Relations Representing the TOR UNIONS

PHONES(PHONE,DEPTNO,LINES)

PHONE	DEPTNO	LINES
2856010	1	800101 1
2856040	2	800101 1 810304 3
2856090	2	811001 2
2856110	3	800101 1
2856000	4	800101 1
2856100	4	800401 1

Table A-15: The Internal View of the TOR PHONES

PHONES(ATTRIBUTE,PTYPE,LTYPE)

ATTRIBUTE	PTYPE	LTYPE
PHONE	I4	1
DEPTNO	I2	2
LINES	I2	3

PHONES1(PHONE,DEPTNO)

PHONE	DEPTNO
2856010	1
2856040	2
2856090	2
2856110	3
2856000	4
2856100	4

PHONES2(PHONE,TIME,LINES)

PHONE	TIME	LINES
2856010	800101	1
2856040	800101	1
2856040	810304	3
2856090	811001	2
2856110	800101	1
2856000	800101	1
2856100	800401	1

Table A-16: The Regular Relations Representing the TOR PHONES

PROJECTS(PROJNO,PROJNM,COST,DEPTNO)

PROJNO	PROJNM	COST	DEPTNO
1000	MDA	800805 5000	800805 4
		811020 7000	801220 2
		820310 9000	811015 3
			821210 1
1010	AI	810201 13000	810201 2
		830405 22000	830310 3
			840107 1
1020	TDMS	831015 18000	831015 3
			831206 2
			850420 1

Table A-17: The Internal View of the TOR PROJECTS

PROJECTS(ATTRIBUTE,PTYPE,LTYPE)

ATTRIBUTE	PTYPE	LTYPE
PROJNO	I2	1
PROJNM	C20	2
COST	F4	3
DEPTNO	I2	3

PROJECTS1(PROJNO,PROJNM)

PROJNO	PROJNM
1000	MDA
1010	AI
1020	TDMS

PROJECTS2(PROJNO,TIME,COST)

PROJNO	TIME	COST
1000	800805	5000
1000	811020	7000
1000	820310	9000
1010	810201	13000
1010	830405	22000
1020	831015	18000

PROJECTS3(PROJNO,TIME,DEPTNO)

PROJNO	TIME	DEPTNO
1000	800805	4
1000	801220	2
1000	811015	3
1000	821210	1
1010	810201	2
1010	830310	3
1010	840107	1
1020	831015	3
1020	831206	2
1020	850420	1

Table A-18: The Regular Relations Representing the TOR PROJECTS

References

[Ackoff 81] Ackoff, R.L.
Creating The Corporate Future.
John Wiley and Sons, New York, N.Y., 1981.

[Ariav 81] Ariav, G., and Morgan, H.L.
MDM: Handling the Time Dimension in Generalized DBMS.
Technical Report DS-WP 81-05-06, Decision Sciences Dept., Univ. of Penn., May, 1981.

[Ariav 83a] Ariav, G.
Preserving the Time Dimension in Information Systems.
Technical Report DS-WP 83-12-06, Decision Sciences Dept., Univ. of Penn., December, 1983.
(Ph.D. Dissertation).

[Ariav 83b] Ariav, G., Clifford, J., and Jarke, M.
Time and Databases.
In ACM-SIGMOD International Conference on Management of Data, pages 243-245. May, 1983.

[Ariav 83c] Ariav, G., Beller, A., and Morgan, H.L.
A Temporal Model.
Technical Report DS-WP 82-12-05, Decision Sciences Dept., Univ. of Penn., March, 1983.

[Ariav 84] Ariav, G., Clifford, J., and Shiftan, J.
A Framework for Implementing Temporally Oriented Databases.
Technical Report , Dept. of Computer Applic. and Info. Sys., New York Univ., February, 1984.

[Ariav 85] Ariav, G.
A Temporally Oriented Data Model.
Technical Report, New York University, GBA-CAIS, March, 1985.

[Astrahan 75] Astrahan, M.M., and Chamberlin, D.D.
Implementation of a Structured English Query Language.
Comm. ACM 18(10):580-587, October, 1975.

[Ben-Zvi 82] Ben-Zvi, J.
The Time Relational Model.
PhD thesis, Dept. of Computer Science, University of California, Los Angeles, 1982.
(Unpublished).

[Bolour 82] Bolour, A., Anderson, T.L., Deketser, L.J., Wong, H.K.T.
The Role of Time in Information Processing: A Survey.
ACM SIGMOD Record (Spring):28-48, 1982.

[Bubenko 77] Bubenko, J.A.
The Temporal Dimension in Information Modeling.
In G.M. Nijssen (editor), Architecture and Models in Data Base Management Systems, pages 93-118. North Holland Publishing Company, Amsterdam, The Netherlands, 1977.

[Chen 76] Chen, P.P.S.
The Entity-Relationship Model: Towards a Unified View of Data.
ACM Trans. on Database Syst. 1(1):9-36, March, 1976.

[Chi 82] Chi, C.S.
Advances in Computer Mass Storage Technology.
Computer 15(5):60-74, May, 1982.

[Clifford 82a] Clifford, J.
A Logical Framework for the Temporal Semantics and Natural-Language Querying of Historical Databases.
PhD thesis, Dept. of Computer Science, SUNY at Stony Brook, December, 1982.

[Clifford 82b] Clifford, J.
A Model for Historical Databases.
In Proceedings of Logical Bases for Data Bases, pages 45-63. Toulouse, France, December, 1982.

[Clifford 83a] Clifford, J., and Warren D.S.
Formal Semantics for Time in Databases.
ACM Trans. on Database Systems 6(2):123-147, June, 1983.

[Clifford 83b] Clifford, J.
Towards an Algebra of Historical Relational Databases.
Technical Report, Computer Applications and Information Systems, New York Univ., December, 1983.

[Clifford 85a] Clifford, J.
Towards an Algebra of Historical Relational Databases.
Proceedings of ACM SIGMOD, Austin Texas :247-265, May, 1985.

[Clifford 85b] Clifford, J.
Towards an Algebra of Historical Relational Databases.
In ACM-SIGMOD Conference on Databases, Austin Texas. May, 1985.

[Codd 70] Codd, E.F.
A Relational Model of Data For Large Shared Data Banks.
Comm. of the ACM 13(6):377-387, June, 1970.

[Codd 74] Codd, E.F.
Recent Investigations in Relational Database Systems.
In ? (editor), Information Processing 74, pages 1017-1021. North-Holland Pub. Co., Amsterdam, 1974.

[Codd 79] Codd, E.F.
Extending the Database Relational Model to Capture More Meaning.
ACM Trans. on Database Syst. 4(4):397-434, December, 1979.

[Copeland 82] Copeland, G.
What if Mass Storage Were Free?
Computer 15(7):27-35, July, 1982.

[Findler 71] Findler, N. and Chen, D.
On The Problem of Time Retrievel, Temporal Relations, Causality, and Coexistance.
In Proceedings of the Second International Joint Conference on Artificial Intelligence, pages ?-?. Imperial College, September, 1971.

[Fries 72] Fries, J.F.
Time-Oriented Patiant Records and a Computer Databank.
Journal of American Medicsal Association 222(12):?-?, December, 1972.

[Gadia 84] Gadia, S. K.
A Homogeneous Relational Model and Query Languages for Temporal Databases.
Technical Report , EE and Computer Science Dept, Texas Tech University, TX 79409, December, 1984.

[Ginzberg 82] Ginzberg, M.J., and Stohr, E.A.
Decision Support Systems: Issues and Perspectives.
In M.J. Ginzberg, W.R. Reitman, and E.A. Stohr (editors), Decision Support Systems, pages 9-32. North-Holland, Amsterdam, 1982.

[Kahn 75] Kahn, K.M.
Mechanization of Temporal Knowledge.
Technical Report MIT-MAC-TR-155, Massachusetts Institute of Technology, Cambridge, Mass., April, 1975.

[Klopprogge 81]
Klopprogge, M.R.
Term: An Approach to Include the Time Dimension in the Entity-Relationship Model.
In P.P.S. Chen (editor), Entity-Relationship Approach to Information Modeling and Analysis, pages 477-512. ER Institute, 1981.

[Lum 84]
Lum, V., et al.
Designing DBMS Support for the Temporal Dimension.
In Proceedings of the ACM SIGMOD Conference, pages 115-126. Boston, June, 1984.

[Maier 82]
Maier, D.
The Theory of Relational Databases.
computer science Press, ?, 1982.

[Mays 81]
Mays, E., Lanka, S., Joshi, A.K., and Weber, B.L.
Natural Language Interaction with Dynamic Knowledge Bases: Monitoring as Responses.
In Proc. of The Seventh International Joint Conference on Artificial Intelligence, pages 134-149. Vancouver, BC, August, 1981.

[Merrett 84]
Merrett T. H.
Relational Information Systems.
Reston Publishing, A Prentice-Hall Company, 1984.

[Morgan 81]
Morgan, H.L.
Decision Support Systems: Technology and Tactics.
In 1981 Digest, pages 307-311. Office Automation Conference, Houston, Tx, 1981.

[Navathe 80]
Navathe, S. B.
Schema Analysis for Database Restructuring.
ACM Trans. on Database Syst. 5(2):157-184, June, 1980.

[Reiter 78]
Reiter, R.
On Closed World Databases.
In Gallaire, H., and Minker, J. (editor), Logic and Databases, pages 55-76. Plenum Press, New York, N.Y., 1978.

[Severance 76]
Severance, D.G., and Lohman, G.M.
Differential Files: Their Application to the Maintenance of Large Databases.
ACM Trans on Database Syst. 1(3):256-267, September, 1976.

[Snodgrass 84] Snodgrass, R.
A Temporal Query Language.
Technical Report, Dept. of Computer Science, UNC, Chapel Hill, NC, ?, 1984.
(Unpublished Report).

[Snodgrass 85] Snodgrass, R.
A Taxonomy of Time in Databases.
Proceedings of ACM SIGMOD, Austin Texas :236-246, March, 1985.

[Sockut 79] Sockut, G.H., and Goldberg, R.
Database Reorganization - Principles and Practice.
ACM Computing Surveys 11(4):371-396, December, 1979.

[Stonebraker 76]
Stonebraker, M.R., Wong, E., and Kreps, P.
The Design and Implementation of INGRES.
ACM Trans. on Database Syst. 1(3):189-222, September, 1976.

[Tsichritzis 82]
Tsichritzis, D.C., and Lochovsky, F.H.
Data Models.
Prentice Hall, Inc., Englewood Cliffs, N.J., 1982.

[Ullman 80] Ullman, J.D.
Principles of Database systems.
Computer Science Press, Potoman, MD, 1980.

[Wiederhold 75]
Wiederhold, G., Fries, J.F., and Weyl, S.
Structured Organization of Clinical Databases.
In Proceedings of the NCC, pages 479-485. AFIPS Press, Montvale, N.J., 1975.