1989

A 2-dimensional temporal relational database model for querying errors and updates, and for achieving zero information-loss

Gautam Bhargava
Iowa State University

INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book. These are also available as one exposure on a standard 35mm slide or as a 17" x 23" black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 9014879

A 2-dimensional temporal relational database model for querying errors and updates, and for achieving zero information-loss

Bhargava, Gautam, Ph.D.

Iowa State University, 1989

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

A 2–dimensional temporal relational database model for querying errors and updates, and for achieving zero information–loss

by

Gautam Bhargava

A Dissertation Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University

Ames, Iowa

1989

TABLE OF CONTENTS

1 INTRODUCTION 1

1.1 Thesis Organization 5

1.2 Preliminary Concepts 6

2 APPROACHES TO TEMPORAL DATABASES: A BRIEF SURVEY 8

3 THE 2–DIMENSIONAL MODEL 13

3.1 The Concept of Time 13

3.2 Instants and Intervals 14

3.3 Temporal Elements 14

3.3.1 Set operations on temporal elements 14

3.3.2 Temporal elements as timestamps 15

3.4 Temporal Assignments 15

3.4.1 Operations on λ–assignments 17

3.4.2 θ–navigation 19

3.5 Temporal Relations 19

3.5.1 Tuples 19

3.5.2 λ–relations 20

3.5.3 Snapshots 20
3.5.4 Weak keys 20
3.5.5 Weak relations 21

4 KEYS FOR HISTORICAL RELATIONS 24

4.1 Historical Relations and Keys 24
4.2 Structure of Weak Relations 25
4.2.1 Restructuring 27

5 ANCHORED RELATIONS 29

5.1 Key for a 2–Relation 29
5.2 Anchors 30
5.2.1 Anchored temporal elements 30
5.2.2 Anchored assignments 31
5.2.3 User operations on anchored assignments 31
5.2.4 Anchored tuples and relations 33
5.2.5 Decomposition of an anchored assignment 34
5.2.6 Extraneous information 34
5.2.7 θ–navigation with anchored assignments 37
5.2.8 Transaction units 37
5.2.9 Query anchors 38
5.2.10 Anchored relations without extraneous information 39

6 HIERARCHY OF USER DOMAINS 41

6.1 User Domains and User Hierarchy 42

6.1.1 Some useful users 42

6.1.2 Extraneous information and keys in different user domains . . 44

7 THE ALGEBRA FOR 2–RELATIONS 48

7.1 Expressions 48

7.1.1 Terms 49

7.1.2 Temporal expressions 52

7.1.3 Boolean expressions 53

7.1.4 Relational expressions 54

7.2 Queries on 2–relations 56

7.2.1 The user domain ■ 57

7.2.2 The user domain ◫ 58

7.2.3 The user domain ⊡ 59

7.2.4 The user domain ◳ 59

7.3 Querying Extraneous Information 61

8 UPDATES IN THE 2–DIMENSIONAL MODEL 63

8.1 Extended Temporal Assignments 64

8.2 Update Operations 65

8.2.1 Some notational conventions 66

8.2.2 Update tags 66

8.2.3 The create update operation 67

8.2.4 The change update operation 69

8.2.5 Redundancy in the change operation 71

8.2.6 The changekey update operation 72

8.3 Update Logs 74
8.3.1 Some definitions 75
8.3.2 Some results 75
8.3.3 Batched updates 76
8.4 Remarks 78

9 QUERYING FOR UPDATES AND ERRORS 79
9.1 Primitives for Querying Updates 79
9.1.1 Update queries and the user hierarchy 80
9.2 Primitives for Querying Errors 81

10 ZERO INFORMATION–LOSS 84
10.1 Components of the Model 85
10.2 Transactions and the Zero Information–Loss Model 86
10.2.1 Definition of the zero information–loss model 86
10.2.2 A comprehensive example 87
10.2.3 Shadow relations 88
10.2.4 Updates 90
10.2.5 Semantics of update operations 92
10.2.6 The append update operation 92
10.2.7 Q–rel 93
10.2.8 Information about a transaction 94
10.2.9 Transactions and transaction log 94
10.3 An Algebra for Zero Information–Loss Model 96
10.3.1 Classical operators for shadow relations 96

10.3.2 Examples 96
10.4 Navigation between $\mathcal{D}$ and *shadow($\mathcal{D}$)* 97
10.4.1 Semijoins of relations in $\mathcal{D}$ and *shadow($\mathcal{D}$)* 98
10.4.2 Filter operations 99
10.5 Operators for Q-Rel 100

11 QL2: A QUERY LANGUAGE FOR THE MODEL 102
11.1 Preliminary Definitions 103
11.2 Basic Queries 105
11.2.1 Syntactic classes in a basic query 106
11.2.2 Procedural semantics of a basic query 107
11.3 Boolean Queries 109
11.4 Other Query Forms 110
11.5 The QL2 Hierarchy 111
11.5.1 The historical user 111
11.5.2 The snapshot user 112
11.6 Queries on Shadow Relations & Q-Rel 112
11.6.1 Queries on shadow relations 112
11.6.2 Queries on Q-rel 116
11.6.3 Examples 117
11.7 QL2 Constructs for Performing Updates 118
11.7.1 The CREATE operation 118
11.7.2 The CHANGE operation 119
11.7.3 The CHANGEKEY operation 120

12 CONCLUSIONS **122**

12.1 Future Directions 123

13 ACKNOWLEDGEMENTS **125**

14 BIBLIOGRAPHY **126**

LIST OF FIGURES

Figure 1.1 The *emp* relation 7

Figure 3.1 Representations of a 2-assignment 18

Figure 3.2 The *emp* 2-relation 22

Figure 3.3 The *management* 2-relation 23

Figure 4.1 Two versions of the *flights* relation 26

Figure 4.2 Snapshot of *flight1*↾NOW 27

Figure 5.1 An anchored assignment 32

Figure 5.2 Decomposition of an anchored assignment 35

Figure 5.3 Value oriented decomposition of an anchored assignment . . . 36

Figure 6.1 The *emp* relation for the $\Box$–user 46

Figure 6.2 The *management* relation for the $\Box$–user 46

Figure 6.3 The *personnel* database for the $\Box$–user 47

Figure 10.1 The model $\langle\langle\{emp\}, \{shadow(emp)\}\rangle$, Q–rel$\rangle$ 89

1 INTRODUCTION

Conventional database systems are capable of storing only the *current* perception of reality and the current relationship among objects. Such databases enforce currency of data by excluding old data values when newer ones become available. After such currency updates, the old values are lost from the logical level and only the current state remains available. On the other hand, *temporal* database systems are capable of storing multiple versions of data, thereby allowing users the facility of examining complete object histories.

Temporal databases commonly store data by associating timestamps with data values. These timestamps denote periods of validity for data values, and by attaching different interpretations to them we arrive at different semantics for data. For example, by construing timestamps as real world time, we can think of temporal data as being the history of some object. In another approach, by construing timestamps as transaction time values, we arrive at the concept of database history of objects.

The relational model can be consistently extended to handle temporal information. However, the complex nature of temporal information results in a more complex model. For example, in the temporal case the concept of keys is very involved, and unlike the traditional case, relational structures are not independent of keys. As a result the concept of keys becomes a central consideration for data modeling.

Most temporal database models in the past have incorporated one dimensional

timestamps. A two dimensional model has been proposed by Snodgrass [27] in which there are valid time (real world time in our terminology) and transaction time dimensions. However, the use of temporal intervals as timestamps, together with an asymmetric treatment of the two time dimensions results in serious limitations in the utility of that model.

This research seeks to extend the temporal model to the case where the timestamps are 2–dimensional, incorporating *both* real world time and transaction time. The relations in our model are called 2–relations. The proposed model incorporates the non-first-normal form (non-1NF) approach, resulting from the use of the more general temporal elements (instead of temporal elements) as timestamps. Additionally, a robust concept for keys provides objects with persistent identities. The result is a powerful framework that allows querying for updates and errors. The model unifies database transactions into the relational approach, and forms the foundation for the zero information-loss model. In our write-up we use the words "object" and "tuple" interchangeably. This is justified by the fact that our non-1NF approach allows the entire existence of an object to be modeled by a single tuple.

In classical relational databases, the concept of an update is external to the model. Information about updates is stored in an ad-hoc manner in transaction logs. In sharp contrast, the 2–dimensional model proposed in this thesis is capable of "storing" updates *within* the relational model. Not only does this permit a formal treatment of updates, but also allows queries to be formulated about the nature of updates. Consequently, it now becomes possible to ask for the effect of any past update at any time.

Since our model is capable of storing multiple versions of history, it is possible to

have queries about the correctness of data. Naturally, this requires that one particular version of data be labeled correct or acceptable. This correct version is used as the basis for comparisons, and is stored as an abstract entity called anchor. Anchors provide data with persistent and correct identities that can be used for identifying objects, as well as with a basis for comparisons in determining errors.

In our model we provide three special update operations. These are more general than the classical update operations — *insert, modify* and *delete.* One single update operation in our model can be used to specify or change (complete) object histories, a task that would require several properly timed operations in the classical model.

Keys for 2–relations play a very important part in the definition and resulting power of our model. Keys are used to provide structure to formless repositories of temporal data known as weak relations [12]. A large part of the complexity of our 2–dimensional model is because of our willingness to accommodate the concept of keys for relations. Not only do keys provide structure to relations, they play a central part in the definition of the relational algebra for our model.

Since keys provide structure, changing from one key to another causes a change in the structure of 2–relations. Such restructuring causes parts of one tuple to migrate to (or combine with parts of) other tuples. However, this migration is not arbitrary but restricted to what are called transaction units. These transaction units correspond to the database history of an object for a given real world instant.

A major contribution of the research reported here is the introduction of the concept of extraneous information. Since a 2–relation can store multiple versions of history for the same object, it can be the case that, for a given instant, one version may have a data value for the object whereas another version might say that the

object is non-existent at that point in time. Information which exists in one version of history but is deemed not to exist in the anchor version is termed extraneous. Such information does not lend itself to restructuring, and we introduce a special relational algebra to handle it.

In our model we introduce the concept of user domains. This is done simply by assigning a user with a time domain which is a subset of the two dimensional temporal universe. The set containment relation, $\subseteq$, among user domains creates a hierarchy of users with different access privileges. At the lowest level of the hierarchy is the user with the zero dimensional time domain, who only sees the classical relational database model and the classical relational algebra. Thus, our model is also a consistent extension of the classical relational model.

The zero information-loss model is another powerful application supported by our 2–dimensional temporal model. In a zero information-loss model we associate a relation *shadow*(r) with every 2–relation r. This relation is used for recording the environmental information associated with an update operation. In addition, a relational structure called Q–rel allows the result of any query to be recomputed at any time in the future. Not only does this allow a user to formulate queries on queries, queries on queries on queries, ad infinitum, but also provides complete information recoverability. Such a zero information-loss model provides a foundation for building secure and auditable database systems.

In order to provide a friendly user interface to the 2–dimensional temporal model and its extensions, we have defined the query language QL2. It is an SQL–based language and provides constructs for the complete 2–dimensional model. Additionally, it has constructs for queries involving the components of the zero information-loss

model. This language supports the user hierarchy, and users at the lowest level are only allowed the use of a subset of this language that corresponds to SQL.

1.1 Thesis Organization

The rest of this thesis is organized as follows. In the rest of this chapter we give some preliminary definitions from the field of relational databases, in order to tie down our terminology. In Chapter 2 we look at the state-of-the-art in temporal databases. In Chapter 3 we define our concepts of two dimensional time, temporal elements, temporal assignments, and 2–dimensional temporal relations. To introduce the concept of keys for temporal relations we provide motivation in Chapter 4 by presenting an example of how key changes affect the structure of temporal relations. However, to avoid distraction we present this example for a historical (or 1–dimensional) temporal relation. Much of the material in Chapters 3 and 4 is drawn from [15]. Using this as a base, we extend the concept to 2–relations in Chapter 5. In the same chapter we identify the need and provide definitions for central concepts such as anchors, extraneous information, and transaction units. This constitutes the definition of our basic model.

In Chapter 6 we look at the user hierarchy, and discuss several different user classes and their role in data retrieval. In Chapter 7 we provide the algebra for our relational model. We define our algebra in two parts — first without considering extraneous information, and then with such information included. In Chapter 8 we define the update operations of our model and provide precise semantics for the same. We prove results that show that any update operation can be re-constructed from the current state of the database. We extend this result to complete update logs and prove

that such logs are an implicit part of the 2-relational model. Having precisely defined our update operations, in Chapter 9 we give the primitives needed for formulating queries on the nature of updates and errors. We extend our relational algebra to include these primitives.

We devote Chapter 10 to the complete account of the zero information-loss model. We provide the concept of shadow relations and Q-rel. We prove the Zero-Loss Theorem that states that a complete transaction log can be recovered from the current state of the zero information-loss model.

In Chapter 11 we provide the syntax and semantics of the query language QL2. In Chapter 12 we present our conclusions.

1.2 Preliminary Concepts

The mathematical concept underlying the relational model [9] is the set-theoretic *relation*, which is a subset of the Cartesian product of a list of domains, where a domain is simply a set of values. The *Cartesian product* of domains D_1, D_2, $\cdots$, D_n, written $D_1 \times D_2 \times \cdots \times D_n$, is the set of all n-tuples $\langle v_1, v_2, \cdots, v_n \rangle$ such that $v_i \in D_i$, for $1 \leq i \leq n$.

A *relation* is any subset of the Cartesian product of one or more domains. In the study of databases, we restrict our attention to finite relations. It helps to view a relation as a table, where each row is a tuple and each column corresponds to one component. The columns are often given names, called *attributes*. The domain of an attribute A is denoted as dom(A). The set of attribute names for a relation is called the *relation scheme*. If r is a relation over the scheme R, we often express this by writing r(R). A *database* is a finite set of relations. The *database scheme* is the set

NAME	SALARY	DEPT
Inga	25K	Clothing
Leu	23K	Toys
Mary	25K	Credit

Figure 1.1: The *emp* relation

of schemes for the relations in the database.

Let τ be a tuple in the relation $r(R)$. Then, for any attribute $A \in R$, $\tau(A)$ represents the value of attribute A for τ. Similarly, for any $X \subseteq R$, $\tau(X)$ represents the values of τ for the attributes in X.

Let $X \subseteq R$ and $Y \subseteq R$ be sets of attributes. Then, X *functionally determines* Y in $r(R)$, written $X \longrightarrow Y$, if $(\forall \tau_1 \forall \tau_2 \in r)(\tau_1(X) = \tau_2(X) \implies \tau_1(Y) = \tau_2(Y))$.

Let r be a relation over the scheme R. A set of attributes $K \subseteq R$ is said to be the *key* for r, if $(\forall \tau_1 \forall \tau_2 \in r)(\tau_1(K) = \tau_2(K) \implies \tau_1 = \tau_2)$.

Example 1.1: Figure 1.1 shows an instance of an *emp* relation, over the scheme NAME, SALARY, DEPT, having NAME as the key.

□

In order to differentiate between the concepts for traditional relational databases and temporal databases, we often qualify the former with the word *snapshot* or *classical.*

2 APPROACHES TO TEMPORAL DATABASES: A BRIEF SURVEY

To quote Richard Snodgrass, "The study of time in databases is currently a 'hot topic'" [26]. In the past few years a large number of approaches for modeling temporal information have been published. In this chapter we present a brief survey of the existing state-of-the-art in temporal databases. Most research in temporal databases has concentrated on developing models that can support powerful data retrieval functions and efficient data organization. Different researchers have considered varying approaches, leading to a number of effective models. Some excellent sources for references to research in temporal databases are [4], [19], and [30]. Here, we look at the important approaches.

Temporal information has been implicitly handled in relational databases for a number of years. Early approaches just modeled time as another attribute domain, without giving it any special semantics. For example, the Query-By-Example system [5] supported both date and time domains. However, because of the simplicity of these approaches it was not possible in such systems to interpret temporal domains properly when computing derived relationships. A greater need for handling temporal domains in an explicit and specialized manner became apparent with the advent of medical information systems such as Time Oriented Databank [34] and CLINFO [23].

In the past decade, temporal database models have received a lot of formal atten-

tion. The work of Clifford and Warren [8] represents one of the first such approaches. In this approach, two special attributes STATE and ?EXISTS are included in every relation scheme. The attribute STATE represents the instant for which the tuple data is recorded. Since a copy of every tuple is included in every state, the attribute ?EXISTS is used to specify whether or not the relationship reported by the tuple exists in the real world for the instant specified by STATE. If a relationship does not exist for a given state, the non-key attributes of the tuple are set to null. This model has an obviously high degree of data redundancy.

Clifford and Warren also formulate a very powerful intensional logic IL_S, a variation of the intentional logic IL of Montague [21] to provide a language for temporal databases. In IL_S the primitive object types are *entity* (an element of the domain of an attribute), *truth* (1 for TRUE and 0 for FALSE), and *state* (an instant of time). A recursive definition allows construction of objects of the complex type $\langle a, b \rangle$, from types a and b. An object of type $\langle a, b \rangle$ is intended to be a function from objects of type a into objects of type b. The logic also allow quantification over variable of any type, yielding a very powerful data retrieval language.

Snodgrass and Ahn [28] introduce a taxonomy of time, and a classification of temporal databases on the basis of the semantics attached to timestamps. In their terminology, interpreting timestamps as the valid time — the real world time period during which data values are valid — leads to *historical databases*. Interpreting time as transaction time — the time at which data values are recorded in the database — leads to *rollback databases*. Historical databases allow us to view the history of an object as it exists in the real world, while rollback databases store different versions of data as it exists in the database. These two time dimensions are orthogonal, and

therefore, it is possible to have a model in which both time dimensions are represented.

The work of Snodgrass [27] incorporates timestamps for data values by adding attributes T_{START} and T_{END} to the schemes of relations. These columns are used to record time values for which data is current. Since time values are stored as separate attributes, the resulting relations have only atomic data values in every column, and thus they are in first normal-form (1NF). This approach causes a lot of data redundancy, since now, a change in one attribute value for a tuple causes the introduction of a whole new tuple in which all the other attribute values are replicated.

Gadia [13] incorporates a new approach for timestamps, using temporal elements instead of the traditional temporal intervals. Temporal elements are formed by taking a union of finitely many intervals. Temporal intervals are closed under union, intersection, and complementation; they form a Boolean algebra. The use of temporal elements greatly simplifies query interfaces to temporal databases [16] and removes much of the data redundancy caused by the 1NF approaches.

The concept of homogeneity of temporal data was first introduced by Gadia in [13]. Informally, this concept requires that the time duration of validity be the same for each attribute within a tuple. This concept is an extension of the assumption in classical databases that requires all the attributes of a tuple to be defined whenever the object represented by the tuple exists. The concept of homogeneity arises naturally and plays an important role in temporal databases. Although this concept was first identified in [13], it is implicitly used by most researchers in temporal databases — Abadi and Manna [1], Clifford and Warren [8], Chomicki and Imielinski [6], Navathe and Ahmed [22], Snodgrass [27], etc.

A number of query languages and relational algebras have been proposed for

information retrieval in temporal databases. A relational algebra and an equivalent tuple calculus for homogeneous temporal relations is given in [13]; a QUEL-like interface for this framework appears in [14]. McKenzie and Snodgrass [20] give an algebra for treating the historical dimension of time; Snodgrass [27] presents a QUEL-based temporal query language, called TQUEL. Navathe and Ahmed [22] have proposed a query language TSQL. Aggregates have appeared in [29], and in [32].

Tansel [31] gives a model and an algebra for temporal databases. His model allows temporal, as well as non-temporal attributes in a relation scheme. The model is non-1NF, but query navigation is done by decomposing tuples into several 1NF tuples. Although a temporal element is not introduced as a data type, the concept is implicit in his work. An *unpack* operation breaks a tuple τ into smaller parts, each of which has an interval timestamp. An attribute in any of these smaller parts can be further decomposed, using operation *t-dec*, into three parts representing the two end points of the interval and the actual data value. This 1NF tuple then can be used for the usual θ–navigation of classical databases. The result tuples are then put back in the non-1NF form by using *t-form* and *pack* operations. Thus, a typical navigation is a five step process: *unpack*, *t-dec*, θ–navigation, *t-form*, and *pack*.

The concept of weak temporal relations and keys for temporal relations was introduced by Gadia [12]. Temporal information in the non-1NF approach is naturally unstructured, and the concept of key plays a vital part in defining the structure of a temporal relation. Informally, two weak temporal relations have the same information content, even though their structures may be different.

A generalized n-dimensional temporal model was suggested by Gadia and Yeung [15]. The special case for $n = 2$, is of tremendous practical use. One potential ap-

plication for this 2-dimensional model was also identified in [15] — namely, providing capability for querying errors made in recording data.

Another model for temporal databases has been developed by Shoshani and Kawagoe [25]. In their approach temporal data is modeled as a time sequence collection represented by a set of triples *(surrogate, time, value)*. Surrogate and time represent coordinates of the value. Operations such as restriction, selection and composition are also defined. Details and extensions of this model are described in [24].

Approaches involving complex objects have also recently been proposed. One such approach by Khoshafian and Copeland [17] proposes a model in which historical data is stored as the pair $\langle interval, value \rangle$. Copeland and Maier's semantic data model approach [10] of labeled sets of heterogeneous values can also be used to represent historical data as $\langle$*time instant, value*$\rangle$, where *time instant* is a transaction time instant. A recent paper by Tansel and Garnett [33] extends nested relations for managing temporal variations of complex objects. In this model, the basic modeling construct is a temporal atom as an attribute value, where a temporal atom is basically what is called a temporal assignment in [13].

3 THE 2–DIMENSIONAL MODEL

3.1 The Concept of Time

In our temporal model, all data values are timestamped with a 2–dimensional time value. Since it is often the case that information being introduced into a database is to remain valid until an unspecified time in the future, we assume we are given a universe of time instants $\mathsf{T}^1 = [0, \infty)$, along with a linear order $\leq$ on it. We assume that T^1 consists of the discrete instants 1, 2, $\cdots$. Such an assumption has been made by Gadia and Vaishnav in [14], and by Tansel in [Clifford and Tansel [7]]. We denote the current time as NOW and assume that $\mathsf{NOW} \in [0, \infty)$. To deal with 2–dimensional timestamps we define $\mathsf{T}^2 = [0, \mathsf{NOW}] \times [0, \infty)$ and require that all timestamps be contained in T^2 . The first dimension of T^2, called *transaction time*, allows us to model the changing knowledge about an object, while the second dimension, called *real world time*, allows us to model the changing history of an object. We introduce several definitions which use elements from one or both dimensions of T^2. Such definitions are often similar; to avoid repetition, we prefix these definitions with λ, where the intended values of λ are 1 and 2.

3.2 Instants and Intervals

An instant is also called a 1–*instant*. If t and t' are 1–instants, $(t,t') \in \mathsf{T}^2$ is called a 2–*instant*. An *interval*, also called a 1–*interval*, is a subset I of [0, NOW] or $[0, \infty)$ such that $\forall\, t_1 t_2 t_3 (t_1 \in I \wedge t_2 \in I \wedge t_1 \leq t_3 \leq t_2 \Rightarrow t_3 \in I)$. If I and J are intervals in [0, NOW] and $[0, \infty)$, respectively, then the set $I \times J \subseteq \mathsf{T}^2$ is called a 2–*interval*.

3.3 Temporal Elements

A λ–*temporal element*, or simply a λ–*element*, is a finite union of λ–intervals. A λ–interval is obviously a λ–element. A 1–instant t may be identified with the 1–element $[t,t]$, and a 2–instant (t,t') may be identified with the 2–element $[t,t] \times [t',t']$; thus, a λ–instant may be regarded as a λ–element.

3.3.1 Set operations on temporal elements

Let μ and ν be λ–elements. Then, the *union* of μ and ν, denoted $\mu + \nu$, is defined as the λ–element $\{t\colon\ t \in \mu \vee t \in \nu\}$; the *intersection*, denoted $\mu * \nu$, is defined as the λ–element $\{t\colon\ t \in \mu \wedge t \in \nu\}$; and, the *difference*, denoted $\mu - \nu$, is defined as the λ–element $\{t\colon\ t \in \mu \wedge t \notin \nu\}$. The *complement* of μ, with respect to the temporal universe T^λ is denoted as $-\mu$. The set of all λ–elements is a Boolean algebra under $+$, $*$, and $-$, with $\emptyset$ as its minimum and T^λ as its maximum element.

We allow set comparisons $\subseteq$ and $=$ among the λ–elements. If μ is a 2–element, then we define $\mho_1(\mu) = \{t$: for some 1–instant t', $(t,t') \in \mu\}$ and $\mho_2(\mu) = \{t'$: for some 1–instant t, $(t,t') \in \mu\}$ as the projections of μ along its first and second

dimensions, respectively. If ν is a 1–element, then $\mu\restriction_1\nu = \mu * (\nu \times [0, \infty))$ and $\mu\restriction_2\nu = \mu * ([0, \text{NOW}] \times \nu)$ are the restrictions of μ to ν in its first and second dimensions, respectively.

Example 3.1: Let μ be the 2–element ([0,10]+[15,25])×[30,45], and let ν be the 1–element [8,40]. Then $\mathfrak{V}_1(\mu)$ = [0,10]+[15,25], and $\mathfrak{V}_2(\mu)$ = [30,45]; the expression $\mu\restriction_1\nu$ = ([8,10]+[15,25]) ×[30,45], and $\mu\restriction_2\nu$ = ([0,10]+[15,25])×[30,40].

3.3.2 Temporal elements as timestamps

In our temporal model, we use temporal elements as timestamps. The use of temporal elements, instead of temporal intervals, provides several advantages. First, we no longer have the huge data redundancy that results from the use of intervals. With intervals, change in any one attribute value causes a new tuple to be introduced. This new tuple has new timestamps for all of the attributes, but only one attribute *value* is new. Therefore, there is a tremendous replication of attribute values, which can lead to serious data redundancy. Second, the use of temporal elements allows natural language queries to be translated very naturally to a database query language. This is because natural language constructs such as "and", "or", and "not" are easily translated into the operations of union, intersection, and complementation of temporal elements.

3.4 Temporal Assignments

To capture the time-variant properties of objects, we introduce the notion of a temporal assignment. A *λ–temporal assignment*, or simply a *λ–assignment*, ξ to an

attribute A, with a λ–element μ as its *temporal domain*, is a function from μ, such that:

1. For each λ–instant $t \in \mu$, $\xi(t)$ is an element of dom(A).

2. The range of ξ is finite, and

3. $\forall a \in \text{dom(A)}$, $\{t : \xi(t) = a\}$ is a λ–element.

We note that not every function from T^λ is a λ–assignment. For example, the function ξ, such that $\xi(t) = a$ if t is even, and $\xi(t) = b$ if t is odd, is not a λ–assignment, since it violates condition 3, above.

If ξ is a λ–assignment, then $[\![\xi]\!]$ denotes its temporal domain and $|\xi|$ denotes its range $\{\xi(t) : t \in [\![\xi]\!]\}$.

For λ–assignments ξ_1 and ξ_2, $\xi_1 \subseteq \xi_2$ holds if $(\forall t \in [\![\xi_1]\!])(\xi_1(t) = \xi_2(t))$.

If ξ is a λ–assignment and ν is a λ–element, then $\xi\restriction\nu$ denotes the restriction of ξ to $\nu * [\![\xi]\!]$ as a function; $\xi\restriction\nu$ is also a λ–assignment. If ξ is a 2–assignment, and μ a 1–element, then $\xi\restriction_1\mu = \xi\restriction(\mu \times [0, \infty))$ and $\xi\restriction_2\mu = \xi\restriction([0, \infty) \times \mu)$. Thus, $\restriction_1$ and $\restriction_2$ restrict a 2–assignment to its first and second dimensions, respectively.

If ξ is a 2–assignment and t is an arbitrary (but fixed) 1–instant, then $\pi_1^t\xi$ is a 1–assignment defined as $\pi_1^t\xi(t') = \xi(t', t)$; and, $\pi_2^t\xi$ is a 1–assignment defined as $\pi_2^t\xi(t') = \xi(t, t')$.

If ξ is a 1–assignment and μ a 1–element, then $\mu{\times}\xi$ is the 2–assignment defined as $(\mu{\times}\xi)(t, t') = \xi(t')$ for $(t, t') \in \mu{\times}[\![\xi]\!]$. Similarly, $\xi{\times}\mu$ is the 2–assignment defined as $(\xi{\times}\mu)(t, t') = \xi(t)$ for $(t, t') \in [\![\xi]\!]{\times}\mu$.

We may denote a λ–assignment ξ as $\langle \nu_1 \mapsto a_1, \cdots, \nu_m \mapsto a_m \rangle$, where $\nu_1, \cdots, \nu_m$ are λ–elements, and $\xi(t) = a_i$ if $t \in \nu_i$, $1 \leq i \leq m$. We also use tabular and pictorial

representation for a 2–assignment, as illustrated in the following example. Note that we use the symbol $\rightsquigarrow$ to stand for NOW in the transaction time dimension.

Example 3.2: Figure 3.1 shows the tabular and pictorial representations of the 2–assignment $\xi = \langle [10,20]\times[0,\infty) \mapsto a,\ [21,\mathsf{NOW}]\times([0,10] \mapsto a, [11,20] \mapsto b)\rangle$. In the pictorial representation, a 1–instant t is represented as [t,t+1), and the region $-[\![\xi]\!]$ is marked with the symbol $\perp$.

3.4.1 Operations on λ–assignments

We say that two temporal assignments ξ_1 and ξ_2 *agree* if

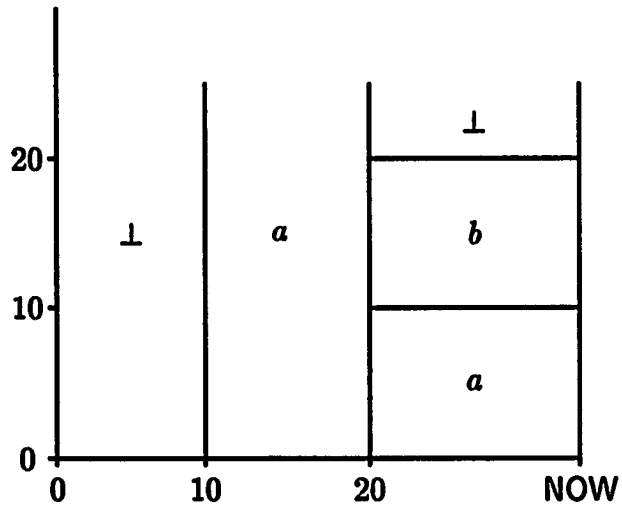
1. They are λ–assignments to the same attribute, and
2. $\forall t \in [\![\xi_1]\!] * [\![\xi_2]\!], \xi_1(t) = \xi_2(t)$.

If ξ_1 and ξ_2 agree, then $\xi_1 \cup \xi_2$ is the common extension of ξ_1 and ξ_2 on $[\![\xi_1]\!]+[\![\xi_2]\!]$; if ξ_1 and ξ_2 do not agree, $\xi_1 \cup \xi_2$ is undefined. If ξ_1 and ξ_2 are λ–assignments to A and $\mu = \{t \in [\![\xi_1]\!] * [\![\xi_2]\!] : \xi_1(t) = \xi_2(t)\}$, then we define $\xi_1 \cap \xi_2$ to be $\xi_1 \restriction \mu$ (or $\xi_2 \restriction \mu$), and $\xi_1 - \xi_2$ to be $\xi_1 \restriction ([\![\xi_1]\!] - \mu)$.

Example 3.3: Suppose A and B are attributes such that dom(A) = dom(B) = {a,b,c}. Further, assume that dom(A) × dom(B) is equipped with <, a transitive relation satisfying a<b<c. Suppose $\xi_1 = \langle [0,5] + [9,9] \mapsto c, [6,7] \mapsto a\rangle$ and $\xi_2 = \langle [3,9] \mapsto a\rangle$ are 1–assignments to A, and $\xi_3 = \langle [8,10] \mapsto c\rangle$ is a 1–assignment to B. Then, $\xi_1 \restriction [7,10] = \langle [7,7] \mapsto a, [9,9] \mapsto c\rangle$. $\xi_1 \cup \xi_2$ is undefined because ξ_1 and ξ_2 take conflicting values during ([3,5]+[9,9]); $\xi_1 \cup \xi_3$ is undefined because ξ_1 and ξ_3 are assignments to different attributes. $\xi_1 \cap \xi_2 = \langle [6,7] \mapsto a\rangle$ and $\xi_1 - \xi_2 = \langle [0,5] + [9,9] \mapsto c\rangle$.

[10,20]	$\times[0, \infty)$	a
[21,$\rightsquigarrow$]	$\times[\ 0,10]$	a
	$\times[11,20]$	b

(a) Tabular Representation



(b) Pictorial Representation

Figure 3.1: Representations of a 2–assignment

3.4.2 θ–navigation

We assume that every attribute is $=$ and $\neq$ comparable. Suppose ξ_1 and ξ_2 are λ–assignments to θ–comparable attributes. We define $[\![\xi_1\ \theta\ \xi_2]\!] = \{t \in [\![\xi_1]\!] * [\![\xi_2]\!] : \xi_1(t)\ \theta\ \xi_2(t)\}$. The construct $[\![\xi_1\ \theta\xi_2]\!]$ is of fundamental importance in temporal databases. It evaluates to a time domain which lies between $\emptyset$ and $[\![\xi_1]\!] * [\![\xi_2]\!]$. If the value is $\emptyset$, it implies that the two λ–assignments were never related. We may use this construct to define $\xi_1\theta_S\xi_2 \leftrightarrow ([\![\xi_1\theta\xi_2]\!] \neq \emptyset)$, and $\xi_1\theta_A\xi_2 \leftrightarrow ([\![\xi_1\theta\xi_2]\!] = [\![\xi_1]\!] * [\![\xi_2]\!] \wedge [\![\xi_1]\!] * [\![\xi_2]\!] \neq \emptyset)$. These two expressions allow us to model the semantics of the words "sometimes" and "always" of natural language.

3.5 Temporal Relations

3.5.1 Tuples

A λ–*tuple* τ, over a scheme R, is a function from R such that for each attribute A$\in$R, τ(A) is a λ–assignment to A. If τ is a λ–tuple over R, and μ a λ–element, then $\tau\restriction\mu$ is the function from R such that $(\tau\restriction\mu)(A) = \tau(A)\restriction\mu$, for every A$\in$R. If τ is a 2–tuple and ν a 1–element, then $\tau\restriction_1\nu$ and $\tau\restriction_2\nu$ are defined in a similar manner. For λ–tuples τ_1 and τ_2 over R, $\tau_1 \subseteq \tau_2$ holds if and only if $\tau_1(A) = \tau_2(A)$ for all A$\in$R.

A λ–tuple τ over R is said to be *homogeneous* if $[\![\tau(A)]\!]$ is the same for all attributes A$\in$R. We will only be interested in homogeneous tuples. A tuple τ is said to be *null* if $[\![\tau(A)]\!] = \emptyset$. For a homogeneous tuple τ over the scheme R, we define its temporal domain $[\![\tau]\!]$ as $[\![\tau(A)]\!]$for any A$\in$R.

3.5.2 λ–relations

A λ–*relation* over the scheme R, is a finite set of non-null homogeneous λ–tuples over R. If r is a λ–relation over R, then$[\![r]\!] = \bigcup_{\tau \in r} [\![\tau]\!]$, and if μ is a λ–element then $r\restriction\mu = \{\tau\restriction\mu : \tau \in r\}$. If $\lambda = 2$, μ is a 1–element, and t is a 1–instant, then $r\restriction_1\mu$, $r\restriction_2\mu$, $\pi_1^t(r)$, and $\pi_2^t(r)$ are defined in a similar manner.

A λ–*database* over a database scheme $R = \{R_1, R_2, \cdots, R_n\}$ is a finite set $\{r_1, r_2, \cdots, r_n\}$ such that r_i is a λ–relation over R_i, for $1 \leq i \leq n$.

3.5.3 Snapshots

If t is a λ–instant, then $r\restriction t$ is called a *temporal snapshot* of r at t. Every assignment in $r\restriction t$ has t as its time stamp, and by omitting it we obtain a snapshot relation over R, called the *static snapshot* of r at t. By a *snapshot* we mean a temporal snapshot or a static snapshot, depending on the context.

3.5.4 Weak keys

Let r be a λ–relation. We say that the *functional dependency* $X \rightarrow Y$ holds in r if it holds in every snapshot of r. If $K \subseteq R$, we say that K is a *weak key* of r if the functional dependency $K \rightarrow R$ holds in r.

Example 3.4: Throughout this thesis we will consider an *emp* 2–relation over NAME SALARY DEPT with NAME as its weak key, and a *management* 2–relation over DEPT MANAGER with DEPT as its weak key. The database {*emp*, *management*} is called the *personnel* database, and Figures 3.2 and 3.3 show a state of the database.

3.5.5 Weak relations

Two λ–relations r and s are said to be *weakly equal*, written $r \sim s$, if for every λ–instant t, $r\restriction t = s\restriction t$. Weak equality is an equivalence relation, and the equivalence class arising from a λ–relation r, denoted $[r]$ is called a λ–*weak relation*. Weak relations and operators for them have been studied in [13].

In going from a relation r to the weak relation $[r]$, we lose the identity of objects in r. Thus, the purpose of a key for λ–relations is to provide a *persistent* identity to objects. For $\lambda = 1$, the concept of keys is quite straightforward. However, for $\lambda = 2$, the concept of objects, and their identities (keys) requires a substantial extension of the basic 2–dimensional temporal model. In the next two chapters we look at the concept of keys for λ–relations.

NAME		SALARY		DEPT	
[8,59] × [11,∞)	John	[8,52] × [11,∞)	15K	[8,39] × [11,∞)	Toys
[60,↝] × [11,60]	John	[53,54] × [11,49]	15K	[40,59] × [11,44]	Toys
		[50,∞)	20K	[45, ∞)	Shoes
		[55,59] × [11,49]	15K	[60,↝] × [11,44]	Toys
		[50,54]	20K	[45,60]	Shoes
		[55,∞)	25K		
		[60,↝] × [11,49]	15K		
		[50,54]	20K		
		[55,60]	25K		
[0,11] × [0, ∞)	Tom	[0,11] × [0, ∞)	20K	[0,11] × [0, ∞)	Hardware
[12,12] × [0,20]	Tom	[12,12] × [0,20]	30K	[12,12] × [0,20]	Hardware
[13,↝] × [0,20]		[13,↝] × [0,20]	20K	[13,↝] × [0,20]	Hardware
+	Tom	[41,51]	30K	[41,51]	Clothing
[41,51]					
[12,24] × [71,∞)	Leu	[12,↝] × [71,∞)	25K	[12,↝] × [71,∞)	Clothing
[25,↝] × [71, ∞)	Inga				
[14,24] × [31,∞)	Inga	[14,↝] × [31,∞)	23K	[14,↝] × [31,∞)	Toys
[25,↝] × [31, ∞)	Leu				
[3, 5] × [0, ∞)	Maria	[3, 5] × [0, ∞)	25K	[3, 5] × [0, ∞)	Credit
[6, 8] × [0,44]	Maria	[6, 8] × [0,44]	25K	[6, 8] × [0,44]	Credit
[9,51] × [0,44]		[9,↝] × [0,44]		[9,↝] × [0,44]	
+	Maria	+	25K	+	Credit
[50,∞)		[50,∞)		[50,∞)	
[52,↝] × [0,44]					
+	Mary				
[50,∞)					

Figure 3.2: The *emp* 2-relation

DEPT		MANAGER	
[8,48] × [11,∞)	Toys	[8,42] × [11,∞)	John
[49,⇝] × [11,49]	Toys	[43,44] × [11,44]	John
		[45,48] × [11,44]	John
		[45,∞)	Leu
		[49,⇝] × [11,44]	John
		[45,49]	Leu
[41,46] × [41,∞)	Clothing	[41,44] × [41,∞)	Tom
[47,64] × [41,47]	Clothing	[45,64] × [41,47]	Tom
[65,⇝] × [41,47]		[65,⇝] × [41,47]	Tom
+	Clothing	[71,∞)	Inga
[71,∞)			

Figure 3.3: The *management* 2-relation

4 KEYS FOR HISTORICAL RELATIONS

Weak relations are, basically, formless repositories of temporal information. Information exists in weak relations in identity-less chunks, and only on combining it with a key, do we get well-structured information about objects with identities [15]. In other words, it is the key that converts a weak relation $[r]$ to a temporal relation r. This suggests that the use of different keys with the same weak relation would yield temporal relations with different structures. These ideas are formalized in this chapter.

4.1 Historical Relations and Keys

We assume that the real world time $\mathsf{T}^1 = [0, \infty)$ is the universe of time. Then, a *historical assignment*, and a *historical tuple* are simply a 1–assignment and a 1–tuple, respectively.

If τ and τ' are tuples over a relation scheme R, and $K \subseteq R$, then they are called K–*compatible*, written $\tau \equiv_K \tau'$, if for all $A \in K$, $|\tau(A)|$ and $|\tau'(A)|$ are singletons and equal. We assume that for a given object, the key values do not change with time. Then, a *historical relation* r over R, with $K \subseteq R$ as its key, is a 1–relation such that

1. $|\tau(A)|$ is a singleton for every $A \in K$ and for every $\tau \in r$, and

2. $\tau \equiv_K \tau'$ implies $\tau = \tau'$, for every τ and τ' in r.

4.2 Structure of Weak Relations

In the introduction to this chapter we noted that different keys when combined with the same weak relation, may yield different historical relations. We demonstrate this phenomenon with the following example.

Example 4.1: Consider the *flights* relation for some airline, over the scheme FLIGHT FROM TO DEPARTS, where for a given flight, these attributes give its flight number, its airport of departure, its airport of arrival, and its time of departure, respectively. We assume that the functional dependency FLIGHT → FROM TO DEPARTS holds for this relation scheme. Also, we assume that the airline does not operate two flights that depart from the same airport, to the same destination, and at the same time. Therefore, the functional dependency FROM TO DEPARTS → FLIGHT also holds. Thus, FLIGHT and FROM TO DEPARTS are two orthogonal keys. Figures 4.1(a) and 4.1(b) show historical versions of the *flights* relation having FLIGHT and FROM TO DEPARTS as their respective keys.

These two versions of the *flights* relation have the same temporal information, yet very different structures. In fact, for any 1-instant t, $flights1\restriction t = flights2\restriction t$. Thus, even though *flights1* and *flights2* are not equal, they are weakly equal. Figure 4.2 shows a snapshot of *flights1* (or *flights2*) for t = NOW. For this snapshot relation, both FLIGHT and FROM TO DEPARTS are keys.

□

FLIGHT	FROM	TO	DEPARTS
[11,∞) F261	[11,∞) JFK	[11,∞) ORD	[11,20] 4PM [21,∞) 5PM
[11,∞) F361	[11,∞) JFK	[11,∞) ORD	[11,20] 6PM [21,∞) 4PM
[20,∞) F461	[20,∞) DSM	[20,∞) LAX	[20,∞) 4PM

(a) *flights1* with FLIGHT as the key.

FLIGHT	FROM	TO	DEPARTS
[11,20] F261 [21,∞) F361	[11,∞) JFK	[11,∞) ORD	[11,∞) 4PM
[21,∞) F261	[21,∞) JFK	[21,∞) ORD	[21,∞) 5PM
[11,20] F361	[11,20] JFK	[11,20] ORD	[11,20] 6PM
[20,∞) F461	[20,∞) DSM	[20,∞) LAX	[20,∞) 4PM

(b) *flights2* with FROM TO DEPARTS as the key.

Figure 4.1: Two versions of the *flights* relation

FLIGHT	FROM	TO	DEPARTS
NOW F261	NOW JFK	NOW ORD	NOW 5PM
NOW F361	NOW JFK	NOW ORD	NOW 4PM
NOW F461	NOW DSM	NOW LAX	NOW 4PM

Figure 4.2: Snapshot of *flight1*↾NOW

4.2.1 Restructuring

The above discussion points to the inherent complexity of temporal information. For temporal relations, the structure of a relation is dependent on the chosen key, whereas in the case of snapshot relations the relation structure is totally independent of the chosen key. A historical relation r can be obtained from a weak relation $[r]$ by combining $[r]$ with a key. Thus, the 1–relation obtained from the weak relation $[r]$, having K as its key, may be denoted as $I_K(r)$, where I_K is called the *restructuring* or *weak identity* operator.

The following theorem formalizes our discussion by stating that the concept of a weak key is adequate for historical relations.

Theorem 1: Let r be a 1–relation over the scheme R. Then K$\subseteq$R is a weak key for r if and only if there is a unique historical relation $s \in [r]$ having K as its key.

Proof: Suppose K is the weak key for r. Then, by definition, K $\rightarrow$ R holds in every temporal snapshot of R. From these temporal snapshots, form new 1–tuples by "pasting" together *all* tuples that have the same values for the attributes in K. The resulting set of 1–tuples, s, has K as its key; s is also weakly equivalent to r, i.e.,

$s \in [r]$.

Conversely, if the 1–relation $s \in [r]$ has K as its key, then $K \rightarrow R$ holds in every snapshot of s. Thus K is also a weak key for s.

□

5 ANCHORED RELATIONS

Temporal information is inherently more complex to model than ordinary snapshot information. As proof, we observed the dependence of the structure of a temporal relation on the chosen key for the relation. The *flights* example in the previous chapter shows a manifestation of this phenomenon for historical relations. In this chapter we examine the effects of keys and restructuring on 2–relations.

5.1 Key for a 2–Relation

For historical relations we made the assumption that the key value of an object does not change with time. However, when we consider both the real world and transaction time dimensions, we have to allow for the possibility that a key value may be recorded erroneously at some time, and that later this value may be modified to reflect the correct value. Therefore, now we assume that even though the key attribute values of an object do not change with time in the real world, such values may sometimes be *recorded* in error. This relaxed assumption adds another level of complexity to the modeling of 2–relations.

Suppose r is a 2–relation over the scheme R. Then $K \subseteq R$ is called the *key* of r provided K is the key of the historical relation $\pi_2^t(r)$ for every (transaction time) 1–instant t in $\mho_1[\![r]\!]$. In other words, K is the key of r if the functional dependency

K⟶R holds in every historical relation $r\restriction t$.

If K is the key for r, then for every $\tau \in r$ and for every A ∈ K, $|\tau(A)\restriction_1 t|$ is a singleton for every transaction time instant t. However, this does not guarantee that $|\tau(A)|$ is a singleton. This means that a key value attribute may no longer be sufficient for identifying an object unambiguously. Furthermore, this means that we can no longer restore a relation r from the weak relation $[r]$.

To resolve the problems mentioned in the previous paragraph, we have to extend the 2–dimensional model. Our extension of the 2–dimensional model is based on the concept of an *anchor*. Informally, an anchor is the true identity of an object. A 2–assignment ξ encodes a version of history for every transaction instant. The version of history that is believed to be correct is designated the anchor, and this anchor value is attached to the assignment itself.

Before formalizing these ideas, we need to introduce the definitions of anchored temporal elements, and anchored temporal assignments. In our definitions we talk about only one attribute (column) of the relation scheme R. This is because different attributes in R have no interaction, and an obvious generalization can be made to include any number of attributes.

5.2 Anchors

5.2.1 Anchored temporal elements

First, we extend the transaction time dimension to include an *abstract* instant, denoted −1. The extended universe for the transaction time dimension is [0, NOW]+ {−1}, which for the sake of convenience is denoted as [−1, NOW].

The set theoretic operators on temporal elements — $+$, $*$, $-$, $\mho_1, \mho_2$, $\vdash$, $\vdash_1$, $\vdash_2$ — can be extended to subsets of $[-1, \mathsf{NOW}]\times[0, \infty)$in an obvious manner.

An *anchored temporal element* is a set of the form $\mu + \{-1\}\times\nu$, where μ is a 2-element, and ν is a 1-element. If μ is a 2-element, then μ^+ denotes the anchored temporal element $\mu + \{-1\}\times\mho_2(\mu)$. μ^+ is called the *anchored extension* of μ. For the sake of convenience, we refer to $\{-1\} \times[0, \infty)$as T^{-1}.

The operators π_1^t, π_2^t, $\vdash, \vdash_1$, $\vdash_2$ are extended to functions from anchored temporal elements in an obvious manner. The operator π_2^{-1} is a special case of π_2^t for $t = -1$.

5.2.2 Anchored assignments

An *anchored assignment* ξ to an attribute A, with μ as its temporal domain, is a function from μ such that $\xi\vdash(\mu * \mathsf{T}^2)$ is a 2-assignment and $\pi_2^{-1}(\xi)$ is a 1-assignment to A.

Example 5.1: Figure 5.1 shows the pictorial representation of the anchored assignment $\langle\{-1\}\times([0,10] \mapsto a,\ [10,20] \mapsto b),\ [10,20]\times[0, \infty) \mapsto a,\ [21, \mathsf{NOW}]\times([0,10] \mapsto a,\ [11,20] \mapsto b)\rangle$.

□

In a later section we will present details how the concept of an anchor is incorporated into a database at the time it is being queried.

5.2.3 User operations on anchored assignments

Recall, $\{-1\}$ is an abstract instant in the transaction time dimension. Its purpose is to "store" the anchor value for an assignment. We do not want a user to obtain the

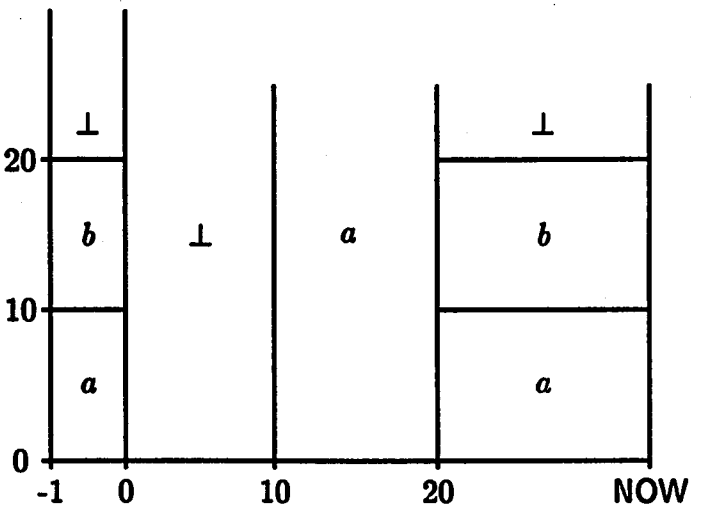


Figure 5.1: An anchored assignment

value $\{-1\}$ during the computation of any temporal element, nor do we want it used as a constant (transaction instant) in a query. Therefore, we allow only restricted access to this abstract entity.

Suppose ξ is an anchored 2-assignment. We define the *restricted temporal domain* of ξ, denoted $[\![\xi]\!]^-$, as $[\![\xi]\!] * \mathsf{T}^2$. Note, that for a 2-assignment ξ_1 without an anchor, $[\![\xi_1]\!]^-$ is the same as $[\![\xi_1]\!]$. If τ is an anchored tuple, and r is an anchored relation, then $[\![\tau]\!]^-$ and $[\![r]\!]^-$ are defined in a natural manner.

Operations such as $\restriction$, $\restriction_1$, etc., can destroy the anchor in an assignment. This is illustrated in the following example.

Example 5.2: Let ξ be the 2-assignment $\langle\{-1\}\times[0,\infty) \mapsto$ John, $[0,10]\times[0,\infty) \mapsto$ Tom, $[11,\mathsf{NOW}]\times[0,\infty) \mapsto$ John$\rangle$. Here, $\langle\{-1\}\times[0,\infty) \mapsto$ John$\rangle$ is the anchor. If μ is the temporal element $[6,8]\times[10,20]$ then $\xi\restriction\mu$ is the assignment $\langle[6,8]\times[10,20] \mapsto$

Tom). Thus, the fact that the "real" identity of this assignment is in fact "John" is now lost. Certainly, we would like to prevent such destruction of identities.

□

To prevent operations such as $\restriction$, $\restriction_1$, etc., from destroying the anchor in an assignment, we redefine them as follows. Let ξ be an anchored 2–assignment, μ a 2–element from T^2, and ν a 1–element. Then $\xi\restriction'\mu$ is defined to be $\xi\restriction\mu^+$; $\xi\restriction'_1$ is defined to be $\xi\restriction_1(\nu + \{-1\})$; and, $\xi\restriction'_2\nu$ is the same as $\xi\restriction_2\nu$.

For an anchored 2–assignment ξ, we define the *anchor of* ξ, denoted $\mathcal{A}(\xi)$, as $\pi_2^{-1}(\xi)$. $\mathcal{A}(\xi)$ is the true identity of ξ and is available for use in queries. This is particularly useful when ξ is an assignment for a key attribute, as it allows an object to be identified unambiguously.

5.2.4 Anchored tuples and relations

An *anchored tuple* τ over R is a function from R such that for every $A \in R$, $\tau(A)$ is an anchored assignment to A. An anchored tuple τ is said to be *homogeneous* if $[\![\tau(A)]\!]$ is the same for all $A \in R$; it is said to be *non-null* if for all $A \in R$ $[\![\tau(A)]\!] \neq \emptyset$.

Suppose τ and τ' are anchored tuples over the scheme R, and $K \subseteq R$. Then τ and τ' are said to be K–*compatible*, written $\tau \equiv_K \tau'$, if for all $A \in K$, $|\tau(A)\restriction_1\{-1\}|$ and $|\tau'(A)\restriction_1\{-1\}|$ are singletons and equal. In other words, τ and τ' are K–compatible if their anchor values for each of the attributes in K are singleton and the same.

If $\tau \equiv_K \tau'$, and $\tau(A)$ and $\tau'(A)$ agree for all $A \in R$, then $\tau \cup \tau'$ is a tuple over R such that $(\tau \cup \tau')(A) = \tau(A) \cup \tau'(A)$, for all $A \in R$.

An *anchored relation* r over the scheme R, with $K \subseteq R$ as its key, is a finite set of non-null, homogeneous tuples over R, such that K is the key of the historical

relation $\pi_2^t(r)$ for every transaction time instant t in $\mathcal{V}_1[\![r]\!]$. The definitions for a *weak anchored relation*, and the *weak equality* of two anchored relations are extended in a natural manner.

5.2.5 Decomposition of an anchored assignment

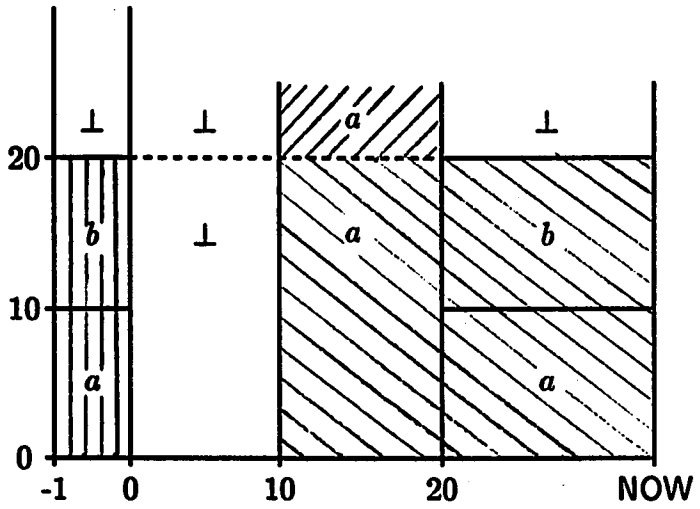
Let ξ be an anchored 2–assignment. Then the 1–element $[\![\mathcal{A}(\xi)]\!]$ is called the *extent* of ξ. If we denote the extent of ξ by ν, then ξ can be partitioned into three parts:

1. $\mathcal{A}(\xi)$ — the anchor of ξ.
2. $\mathcal{S}(\xi)$ — the *shielded* portion of ξ, defined as $\xi\restriction([0, \mathsf{NOW}]\times\nu)$.
3. $\mathcal{X}(\xi)$ — the *extraneous* portion of ξ, defined as $\xi\restriction([0, \mathsf{NOW}] \times ([0, \infty) - \nu))$.

Clearly, ξ is the disjoint union of $\{\text{-}1\}\times\mathcal{A}(\xi)$, $\mathcal{S}(\xi)$, and $\mathcal{X}(\xi)$. Figure 5.2 shows the decomposition of the anchored assignment of Figure 5.1.

5.2.6 Extraneous information

In the extended 2–dimensional model, the anchor is used to designate some version of the stored history of an object as the "true" version. The choice of an anchor, though arbitrary, is not made at random. It is guided by the user's (or the database administrator's) perception of reality. This choice, however, can cause the true version to say that a particular database object does not exist for some real world instant, even though some other version may have data about the object for the very same real world instant. This leads to the concept of extraneous information.



Legend: $\mathcal{A}(\xi)$; $\mathcal{S}(\xi)$; $\mathcal{X}(\xi)$.

Figure 5.2: Decomposition of an anchored assignment

Extraneous information is the information about those real world instants when an object, or the relationship among objects, does not exist according to the accepted, or anchor, version of reality; however, according to some other (non-anchor) version, some information does exist for the *same* real world instant. The extraneous information in an assignment ξ is the part of the anchor denoted $\mathcal{X}(\xi)$.

In the presence of extraneous information, 2–relations cannot be restored from weak relations by just using a key. Intuitively, this is because extraneous information does not have an identity and, therefore, when restructuring we cannot designate it as part of any object. As an example of this problem, consider the following scenario.

Suppose that the anchored assignment of Figure 5.1 corresponds to attribute A for some object in relation r. Then, since there are transaction instants when its real

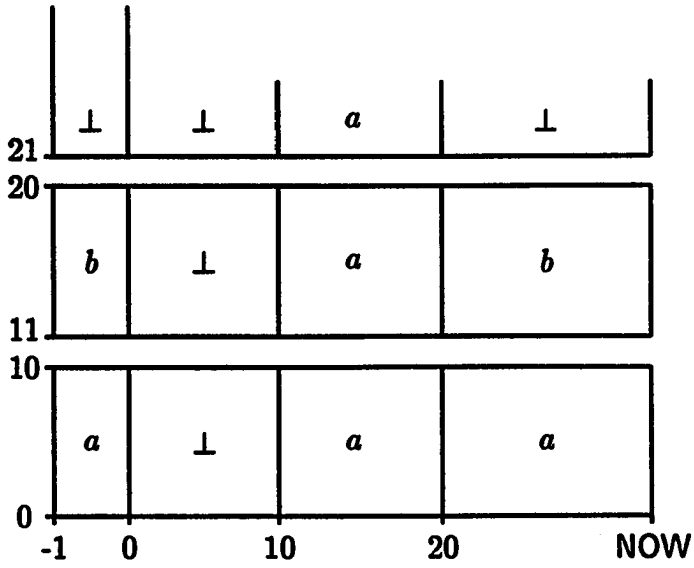


Figure 5.3: Value oriented decomposition of an anchored assignment

world history takes on multiple values, by definition, A cannot be a key attribute. However, an algebraic operation may cause A to become a key attribute. This could happen, for example, if r is a relation over BA, with B as its key, and we may want to compute $\Pi_A(r)$. To compute $\Pi_A(r)$, we would first compute the relation $r' = \{\tau(A): \tau \in r\}$. Then, to ensure that A was, indeed, the key for r', we would break ξ into several parts based on the values in $\mathcal{A}(\xi)$, the anchor of ξ. (Since we accept the anchor as the true version of history). This would cause ξ to split up into the three parts shown in Figure 5.3. The extraneous part $([10,20]\times[31,\infty) \mapsto a)$ does not have an identity and cannot be included in the result relation r'. This is an acceptable situation since this has been dictated by our accepted version of reality, and in that version this information about ξ does not even exist.

Identity-less chunks, such as the one in Figure 5.3 may arise out of several tuples,

and their role in restructuring is not clear. Such chunks of information cannot be attached to the history of any object having a fixed identity. We will disregard extraneous information until Section 7.3.

Notation: Let ξ be an anchored 2-assignment. Then, we use $\lceil\lceil\xi$ to denote the anchored 2-assignment $(\{-1\}\times\mathcal{A}(\xi)) \cup \mathcal{S}(\xi)$. Thus, $\lceil\lceil\xi$ represents the non-extraneous part of ξ.

5.2.7 θ–navigation with anchored assignments

Let ξ_1 and ξ_2 be anchored assignments. As before, we define $[\![\xi_1\theta\xi_2]\!] = \{(t,t') : \xi_1(t,t')\theta\xi_2(t,t')\}$. Additionally, we also define $[\![\xi_1\theta\xi_2]\!]^- = \{(t,t') : \xi_1(t,t')\theta\xi_2(t,t') \wedge \xi_1(-1,t')\theta\xi_2(-1,t')\}$. Note that the additional requirement $\xi_1(-1,t')\theta\xi_2(-1,t')$ in the definition of $[\![\xi_1\theta\xi_2]\!]^-$ excludes the instants when the θ–relationship between ξ_1 and ξ_2 is deemed not to exist by the accepted version of reality. In other words, this new definition excludes those instants when the computed information is extraneous.

5.2.8 Transaction units

A close examination of the two versions of the *flights* relation of Figure 4.1 reveals that in order to accommodate a change of key, we have to paste certain parts of the real world history of one object to similar parts of other objects, to form objects with new identities. For example the [11,20] part of object "F261" of *flights1* and the [21,∞) part of object "F361" of *flights1* combine to form the object "JFK ORD 4PM" of *flights2*. In order to keep the temporal information meaningful, we only allow certain parts of objects to migrate to (or combine with parts of) other objects. These basic units, called transaction units, cannot be split during restructuring. They

are formally defined below.

Let ξ be an anchored assignment to an attribute, and let t be a real world instant. Then, $\xi\restriction([0, \mathsf{NOW}]\times t)$ is called a *transaction unit* of ξ at the real world instant t. This transaction unit represents the database history of the single value identified by $\xi(-1, t)$. To maintain the integrity of temporal information, we only allow those transaction units to be combined that can be identified as parts of the same object. (Of course, the concept of object is determined by the key being used.)

Two finite sets, r and s, of anchored tuples over a scheme R are defined to be *weakly transaction equivalent* if r and s are weakly equal, and if every transaction unit in r is a transaction unit in s, and vice versa. We will only be interested in weak relations that are transaction equivalent.

5.2.9 Query anchors

In its raw form, temporal information does not have any structure. Keys provide structure to such information. Furthermore, in the case of 2-dimensional temporal relations, one version of data is treated as special — it is the accepted true value. For the purpose of querying, the users of a temporal database system need to identify objects unambiguously. Thus, each user has to be able to refer to the anchor values of objects in the database.

To query a database, a user needs a transaction instant of time, t, called the *query anchoring point*, at which the database is believed to be correct. The query anchoring point is user (or database administrator) specified. The query anchoring point t is used by the database system to implicitly extend assignments in the database. If ξ is a (non-anchored) 2-assignment, then the *t-anchored extension* of ξ, denoted ξ^t, is

the anchored assignment $\xi \cup \{-1\} \times \pi_2^t(\xi)$. Clearly, $\mathcal{A}(\xi^t) = \pi_2^t(\xi)$.

Example 5.3: If $t = 40$, such that $40 <$ NOW, then Figure 5.1 shows the anchored extension of the 2–assignment of Figure 3.1.

□

If τ is a 2–tuple over the scheme R, then the *t–anchored extension* of τ, denoted τ^t is an anchored tuple such that $\tau^t(A)$ is $(\tau(A))^t$ for all $A \in R$. Similarly, if r is a finite set of 2–tuples, then the *t–anchored extension* of r, denoted r^t, is defined to be $\{\tau^t : \tau \in r\}$.

5.2.10 Anchored relations without extraneous information

If τ is an anchored tuple over the scheme R, then $\lceil\lceil\tau^t$ is an anchored tuple over the scheme R such that $(\lceil\lceil\tau^t)(A) = \lceil\lceil\tau^t(A)$ for every $A \in R$. Similarly, if r is a set of anchored tuples over R, then $\lceil\lceil r^t = \{\ \lceil\lceil\tau^t : \tau \in r\} - \{\emptyset\}$.

The following Theorem now holds immediately.

Theorem 2: Let r be a 2–relation over the scheme R, such that $K \subseteq R$, and let t be a query anchoring point. Then $K \rightarrow R$ holds in $\lceil\lceil r^t$ if and only if there is a unique anchored relation $s \in [\lceil\lceil r^t]$ with K as its key, such that s is transaction equivalent to $\lceil\lceil r^t$.

□

This theorem states that anchored relations without extraneous information form equivalence classes of weak relations from which a relation can be obtained by restructuring along a key.

The relation s in the above Theorem is denoted as $I_{\mathrm{K}}(\prod \mathrm{r}^{t})$, and I_{K} is called a *weak identity operator* for anchored relations.

6 HIERARCHY OF USER DOMAINS

The 2-dimensional temporal model allows us to create user domains with varying amounts of access privileges. This results in a hierarchy of users, organized on the access privileges. A user at the top of the hierarchy has access to the entire temporal database, whereas at the lowest level the user can access only a static snapshot of data. These ideas are formalized in this chapter.

Notation: If r and r' are λ-relations over the scheme R, we write $r \sqsubseteq r'$ if for every $\tau \in r$ there is a $\tau' \in r'$ such that $\tau \subseteq \tau'$. By extension, if $\mathcal{D}$ and $\mathcal{D}'$ are λ-databases over the database scheme R, we write $\mathcal{D} \sqsubseteq \mathcal{D}'$ if for every R$\in R$, if $r \in \mathcal{D}$ and $r' \in \mathcal{D}'$ are relation over R then $r \sqsubseteq r'$.

□

We assume that a database $\mathcal{D}$ consisting of anchored 2-relations $r_1, r_2, \cdots, r_n$, is given. Furthermore, we assume that there is no extraneous information in $\mathcal{D}$. (If this is not the case, each relation $r_i \in \mathcal{D}$ can easily be modified to $\lceil\lceil r_i^t$ for some anchoring point t). If μ is a 2-element, then $\mathcal{D}\restriction\mu$ denotes $\{r_1\restriction\mu, r_2\restriction\mu, \cdots, r_n\restriction\mu\}$. Clearly, $\mathcal{D}\restriction\mu \sqsubseteq \mathcal{D}$. We assume that $\mathcal{D}$ is the general purpose database that can be queried by a variety of users.

6.1 User Domains and User Hierarchy

We define a *user domain* $\sqcup$ to be a 2–element in $[0, \mathsf{NOW}]\times[0, \infty)$, and by a $\sqcup$–user we mean a user with user domain $\sqcup$. Only the information in $\mathcal{D}\restriction\sqcup$ is available to a $\sqcup$–user, and an instant in $\mathcal{V}_1(\sqcup)$ can be used as a query anchoring point for this user. Note, if $\mathsf{NOW} \in \mathcal{V}_1(\sqcup)$, it can be used as a query anchoring point, even though it gets a new value every instant.

With these definitions, it follows that the set containment relation $\subseteq$ creates a simple user hierarchy in the sense that a user at an upper level in the user hierarchy can access everything that a user at a lower level can. This is formally stated in the following Theorem.

Theorem 3: If $\mu \subseteq \nu$, then $\mathcal{D}\restriction\mu \sqsubseteq \mathcal{D}\restriction\nu$.

Proof: The proof follows immediately from the fact that $\mu \subseteq \nu$ implies $(\mathcal{D}\restriction\nu)\restriction\mu = \mathcal{D}\restriction\mu$.

□

Convention: For the sake of convenience, we often refer to a $\sqcup$–user simply as $\sqcup$.

6.1.1 Some useful users

In theory it is possible to designate any subset of $[0, \mathsf{NOW}]\times[0, \infty)$ as a user domain. However, when looking at the practical aspects of the model, some of these user domains stand out as very useful.

The *system user* is $[0, \mathsf{NOW}]\times[0, \infty)$, and is denoted by the graphic symbol ■. The whole database is accessible to ■. Such a user can query all aspects of the

database, to be introduced in later chapters.

Certain database applications are concerned only with the actions that have taken place, without any concern for events planned or anticipated in the future. Such a situation suggest the need for a user domain which has access only to events in the interval [0, NOW]×[0, ∞) $*$ $\{(t,t') : t' \leq t\}$. This user has no concept of the future, and by using NOW as the query anchoring point, he can monitor how well records are being kept. We use the graphic symbol ◪ for such a user.

The *historical user*, denoted by the symbol ◫, is of the form $\{t\}\times[0, \infty)$, for some 1–instant t. Such a user has t as the anchoring point, and for every assignment in this user's view the anchor and the shielded parts are the same. As a result, the anchor is of no practical use for this kind of user and it can be disregarded in the algebra for such a user. Moreover, the transaction timestamp for every assignment occurring in $\mathcal{D}$↾◫ is t, and hence it too may be disregarded. Thus, such a user sees the database as a historical database. The historical user NOW×[0, ∞), denoted by the symbol ❒ is particularly useful, becauses he has access to only the most current history of objects.

The *snapshot user*, denoted by the symbol ⊡, is of the form $\{t\}\times\{t'\}$, for some 1–instants t and t'. For the snapshot user all timestamps are $\{t\}\times\{t'\}$, and therefore convey no temporal information. Thus, timestamps may be eliminated for a snapshot user. The snapshot user NOW×NOW, denoted by the symbol ❐, comes closest to the classical database user.

The user $\{(t,t') : t = t'\}$, denoted by the symbol ◰, is interesting because he can see what has been available to classical users, at every instant in the evolution of the database.

There are other interesting users. For example, the user [0,NOW– 10] ×[0, ∞) can only see data which is at least 10 units old. This may be used, for instance, by the government for declassifying old information. The user NOW×[NOW+ 1, ∞) can only see the current knowledge about the future.

6.1.2 Extraneous information and keys in different user domains

The use of anchors can make certain information in an assignment extraneous. This, however, can only happen for users whose domain is 2–dimensional. For historical users, there is only one version of history available, and that is also the anchor version. Therefore, there is no data which exists in one version but does not exist in the accepted or anchor version. In other words, there is no extraneous information. The same is, obviously, true for snapshot users. This idea is formally stated in the following Theorem.

Theorem 4: Suppose $\boxplus$ is the historical user $\{t\}\times[0, \infty)$, $\boxdot$ is the snapshot user $\{t\}\times\{t'\}$, and t is a query anchoring instant. If r is a 2–relation, then $(\lceil\lceil r^t)\restriction\boxplus = r\restriction\boxplus$ and $(\lceil\lceil r^t)\restriction\boxdot = r\restriction\boxdot$.

Proof: Since $\boxplus = \{t\}\times[0, \infty)$, $\lceil\lceil r^t\restriction\boxplus$ only contains the historical version of data recorded at transaction time t. Therefore, an assignment $\xi \in (\lceil\lceil r^t)\restriction\boxplus$ if and only if ξ resides in $r\restriction\boxplus$. Thus $(\lceil\lceil r^t)\restriction\boxplus = r\restriction\boxplus$.

A similar argument proves $(\lceil\lceil r^t)\restriction\boxdot = r\restriction\boxdot$.

□

For a 2–relation without extraneous information, a key can be used for restructuring. The same is true for historical relations. When considering historical user

domains we want the concept of a key to be downward compatible with the concept of a key for the system user, ■. The following theorem states that for a historical user this is true. Further, it states that for a snapshot user the concept of a key is independent of the structure of the relation, thereby mimicing the classical relational model.

Theorem 5: Suppose ◫ is the historical user $\{t\}\times[0, \infty)$, ⊡ is the snapshot user $\{t\}\times\{t'\}$, and t is a query anchoring instant. If r is a 2–relation over the scheme R, and K ⊆ R is a weak key of r, then $(I_K(\lceil\rceil r^t))\restriction◫ = I_K(r\restriction◫)$ and $(I_K(\lceil\rceil r^t))\restriction⊡ = I_K(r\restriction⊡) = r\restriction⊡$.

□

Example 6.1: Consider the *personnel* database of Figures 3.2 and 3.3. A ◫–user may view *emp*$\restriction$◫ and *management*$\restriction$◫ as historical relations with NAME and DEPT as their respective keys. After omitting the transaction time dimension, the current state of the historical database *personnel*$\restriction$◫ is shown in Figures 6.1 and 6.2.

□

Example 6.2: A ⊡–user may view the *emp*$\restriction$⊡ and *management*$\restriction$⊡ relations as snapshot relations with NAME and DEPT as their respective keys. A state of the *personnel* database, as seen by the ⊡–user is shown in Figure 6.3.

□

NAME	SALARY	DEPT
[11,60] John	[11,49] 15K [50,54] 20K [55,60] 25K	[11,44] Toys [45,60] Shoes
[0,20] + Tom [41,51]	[0,20] 20K [41,51] 30K	[0,20] Hardware [41,51] Clothing
[71,∞) Inga	[71,∞) 25K	[71,∞) Clothing
[31,∞) Leu	[31,∞) 23K	[31,∞) Toys
[0,44] + Mary [50,∞)	[0,44] + 25K [50,∞)	[0,44] + Credit [50,∞)

Figure 6.1: The *emp* relation for the □-user

DEPT	MANAGER
[11,49] Toys	[11,44] John [45,49] Leu
[41,47] + Clothing [71,∞)	[41,47] Tom [71,∞) Inga

Figure 6.2: The *management* relation for the □-user

NAME	SALARY	DEPT
Inga	25K	Clothing
Leu	23K	Toys
Mary	25K	Credit

The *emp* relation for the $\square^{\bullet}$–user

DEPT	MANAGER
Clothing	Inga

The *management* relation for the $\square^{\bullet}$–user

Figure 6.3: The *personnel* database for the $\square^{\bullet}$–user

7 THE ALGEBRA FOR 2–RELATIONS

We suppose that a database $\mathcal{D} = \{r_1, r_2, \cdots, r_n\}$, where r_i is a 2–relation over the scheme R_i, with $\mathrm{K}_i \subseteq \mathrm{R}_i$ as its key, $1 \leq i \leq n$, a user domain $\sqcup \subseteq [0, \mathsf{NOW}]\times[0, \infty)$, and a query anchoring point $t \in \mathcal{U}_1(\sqcup)$ are given. Until Section 7.3, we assume that the extraneous information is disregarded; therefore, when a user queries $r \in \mathcal{D}$, the system anchors it at t to obtain r^t, and then interprets it as $\lceil\lceil r^t \rceil'\sqcup$, where $\lceil\lceil r^t$ is obtained from r by deleting the extraneous information in it. In Section 7.3 we will include relations with extraneous information in our algebra. To emphasize the inclusion of extraneous information in relations r, s, etc., we will write them as r^+, s^+, etc.

7.1 Expressions

The set of all algebraic expressions, denoted $\mathcal{E}$ can be divided into four mutually exclusive groups: $\mathcal{T}$, $\mathcal{B}$, $\mathcal{TE}$, and $\mathcal{R}$, consisting of terms, Boolean expressions, temporal expressions, and relational expressions respectively. A *term* is the syntactic counterpart of a temporal assignment. We can write $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$, where elements of $\mathcal{T}_\lambda$ are syntactic counterparts of λ–assignments, for $\lambda = 1, 2$. A *Boolean expression* is the syntactic counterpart of the Boolean truth values TRUE and FALSE. A *temporal expression* is the syntactic counterpart of a temporal element. We can write $\mathcal{TE} =$

$\mathcal{TE}_1 \cup \mathcal{TE}_2$, where $\mathcal{TE}_\lambda$ is the syntactic counterpart of a λ–element, for $\lambda = 1, 2$. Finally, a *relational expression* is the syntactic counterpart of an anchored relation.

Every expression $e \in \mathcal{E}$ is defined over a set of attributes R. If $e \in \mathcal{T} \cup \mathcal{B} \cup \mathcal{TE}$, it is often convenient to think of R as a list of parameters for e, and we write e as $e\langle \mathrm{R} \rangle$. The expression $e\langle \mathrm{R} \rangle$ yields a concrete object $e(\tau)$, when a tuple τ over the scheme $\mathrm{R}' \supseteq \mathrm{R}$ is substituted in it. Note that the set of attributes R may be empty, in which case e yields a concrete object without tuple substitution. If $e \in \mathcal{T} \cup \mathcal{B} \cup \mathcal{TE}$ is defined over R, we also consider it defined over $\mathrm{R}' \supseteq \mathrm{R}$. We define $\mathcal{T}\langle \mathrm{R} \rangle = \{e \in \mathcal{T} : e$ is a term over R$\}$. $\mathcal{B}\langle \mathrm{R} \rangle$ and $\mathcal{TE}\langle \mathrm{R} \rangle$ are defined in a similar manner. It follows that if $\mathrm{R} \subseteq \mathrm{R}'$, then $\mathcal{T}\langle \mathrm{R} \rangle \subseteq \mathcal{T}\langle \mathrm{R}' \rangle$, $\mathcal{B}\langle \mathrm{R} \rangle \subseteq \mathcal{B}\langle \mathrm{R}' \rangle$, and $\mathcal{TE}\langle \mathrm{R} \rangle \subseteq \mathcal{TE}\langle \mathrm{R}' \rangle$.

The set of all expressions $\mathcal{E}$ is defined recursively. For each expression, we also discuss its semantics. For $e \in \mathcal{R} \cup \mathcal{TE}\langle \emptyset \rangle \cup \mathcal{B}\langle \emptyset \rangle$, we define $\mathcal{I}(e)$, the *interpretation* of e. For expressions in $(\mathcal{T} \cup \mathcal{TE} \cup \mathcal{B}) - (\mathcal{TE}\langle \emptyset \rangle \cup \mathcal{B}\langle \emptyset \rangle)$, we define the result of *tuple substitution*. Strictly speaking, for an expression e in $\mathcal{TE}\langle \emptyset \rangle \cup \mathcal{B}\langle \emptyset \rangle$, we need to define the result of tuple substitution also. However, we observe that the result is independent of the tuple being substituted, and that it is the same as $\mathcal{I}(e)$.

Notation: We write $\tau \succeq \mathrm{R}$ to mean that τ is a tuple over a *superset* of the scheme R.

7.1.1 Terms

Terms are the syntactic counterparts of anchored assignments, and tuple substitution in a term yields an anchored assignment for the attribute which is the *post* of the term. (The concept of post is defined in T1–T5 below, and is useful in stating TE4 and BE3 precisely). The set of all terms $\mathcal{T}$ is defined as follows.

T1 A constant 2-assignment ξ to an attribute A, such that $[\![\xi]\!] \subseteq \sqcup$, is a term in $\mathcal{T}_2\langle A\rangle$, with A as its *post*. If τ is a tuple such that $\tau \succeq A$, we define $\xi(\tau) = \xi \cup \{-1\} \times \pi_2^t(\xi)$. Note, that the anchor part in $\xi(\tau)$ is not provided by the user, but is tacked on by the system. In particular, we abbreviate $\xi = \langle \sqcup \mapsto a\rangle$ as a, and thus $a(\tau) = \langle \sqcup^+ \mapsto a\rangle$.

T2 If u is a term in $\mathcal{T}_2\langle R\rangle$ with $A \in R$ as its post, then its anchor, $\mathcal{A}(u)$, is a term in $\mathcal{T}_1\langle A\rangle$ with A as its *post*. We define $\mathcal{A}(u)(\tau) = \mathcal{A}(u(\tau))$. $\mathcal{A}(u)$ allows a user to input information about an anchor, and is particularly useful when A is a key attribute. Clearly, if $a \in \mathrm{dom}(A)$, then $\mathcal{A}(a)(\tau) = \mathcal{A}(\langle \sqcup^+ \mapsto a\rangle) = \langle \mathcal{V}_2(\sqcup) \mapsto a\rangle$.

T3 For each attribute A, $\hat{A} \in \mathcal{T}_2\langle A\rangle$. A is called the *post* of the term $\hat{A}$. If $\tau \succeq A$, $\hat{A}(\tau) = \tau(A) \restriction' \sqcup$. As a notational convenience, we simply write A for the term $\hat{A}$.

T4 If $u_1 \in \mathcal{T}_1\langle R\rangle$ and $u_2 \in \mathcal{T}_2\langle R\rangle$, with A as their post, $t_1 \in \mathcal{TE}_1\langle S\rangle$ and $t_2 \in \mathcal{TE}_2\langle S\rangle$, then $u_1 \restriction' t_1 \in \mathcal{T}_1\langle RS\rangle$, $u_2 \restriction' t_2$, $u_2 \restriction'_1 t_1$, and $u_2 \restriction'_2 t_1 \in \mathcal{T}_2\langle RS\rangle$, and their *post* is A. If $\tau \succeq RS$, then $u_1 \restriction' t_1(\tau) = u_1(\tau) \restriction' t_1(\tau)$, $u_2 \restriction' t_2(\tau) = u_2(\tau) \restriction' t_2(\tau)$, $u_2 \restriction'_1 t_1(\tau) = u_2(\tau) \restriction'_1 t_1(\tau)$, and $u_2 \restriction'_2 t_1(\tau) = u_2(\tau) \restriction'_2 t_1(\tau)$. Recall, the operators $\restriction'$, $\restriction'_1$, and $\restriction'_2$ do not destroy the anchor.

T5 We say that a temporal expression t in $\mathcal{TE}_1\langle S\rangle$ is a *syntactic instant* if t is an instant from T^1, or there exists $t' \in \mathcal{TE}_1\langle S\rangle$, such that t is either $fi(t')$ or t is $li(t')$. If $u \in \mathcal{T}_2\langle R\rangle$ with A as its post, t is a syntactic instant, then $\pi_1^t(u)$ and $\pi_2^t(u)$ are in $\mathcal{T}_1\langle RS\rangle$, with A as their *post*. If $\tau \succeq RS$, then $\pi_1^t(u)(\tau) = \pi_1^{t(\tau)}(u(\tau))$. If $t(\tau) \in [0, \mathsf{NOW}]$, then $\pi_2^t(u)(\tau) = \pi_2^{t(\tau)}(u(\tau))$; otherwise, it is

undefined.

Example 7.1: Consider the *personnel* database of Figures 3.2 and 3.3. Both SALARY and $\mathcal{A}$(SALARY) are terms. (Recall, for convenience we represent a term such as $\widehat{\text{SALARY}}$ simply as SALARY). Suppose τ is John's tuple from the *emp* relation, the query anchoring instant t = NOW, and $\tau' = \lceil\tau^t$. We look at the evaluation of SALARY(τ') for different users. For a ■–user this term evaluates to the assignment

$$\begin{aligned}\xi = \langle &[8,52]\times[11,\infty) \mapsto 15\text{K},\\ &[53,54]\times([11,49] \mapsto 15\text{K},\ [50,\infty) \mapsto 20\text{K}),\\ &[55,59]\times([11,49] \mapsto 15\text{K},\ [50,54] \mapsto 20\text{K},\ [55,\infty) \mapsto 25\text{K}),\\ &[60,\text{NOW}]\times([11,49] \mapsto 15\text{K},\ [50,54] \mapsto 20\text{K},\ [55,60] \mapsto 25\text{K})\rangle.\end{aligned}$$

For the same user, $\mathcal{A}$(SALARY)(τ') evaluates to the 1–assignment

$$\langle[11,49] \mapsto 15\text{K},\ [50,54] \mapsto 20\text{K},\ [55,60] \mapsto 25\text{K}\rangle.$$

The ◻–user views the *personnel* database as the historical database shown in Figures 6.1 and 6.2. For such a user, SALARY(τ') evaluates to the assignment

$$\xi = \langle\{\text{NOW}, -1\}\times([11,49] \mapsto 15\text{K},\ [50,54] \mapsto 20\text{K},\ [55,60] \mapsto 25\text{K})\rangle.$$

The transaction timestamp is always {NOW, −1}, and on excluding it we get the historical assignment $\langle([11,49] \mapsto 15K,\ [50,54] \mapsto 20K,\ [55,60] \mapsto 25K)\rangle$. Thus, the ◻–user gets the impression that SALARY(τ') was directly computed from the historical version of the *emp* relation in Figure 6.1.

Since a NOW×(NOW, ∞)–user only sees the current knowledge about the future, and as τ'(SALARY) ↾ (NOW×(NOW, ∞) = ∅, such a user has no knowledge of John's salary. Similarly, the snapshot user ◻ has no knowledge about John's tuple.

□

7.1.2 Temporal expressions

Temporal expressions are the syntactic counterpart of λ–elements. The set $\mathcal{TE}$ of temporal expressions can be represented as $\mathcal{TE}_1 \cup \mathcal{TE}_2$, where $\mathcal{TE}_\lambda$ is the set of λ-dimensional temporal expressions, for $\lambda = 1, 2$. When a tuple is substituted in a temporal expression, we get a λ–element corresponding to that part of the temporal domain of the tuple which "satisfies" the temporal expression. The set of all temporal expressions is recursively defined as follows.

TE1 Every temporal element $\mu \in \sqcup$ is in $\mathcal{TE}_2\langle\emptyset\rangle$. Also, every 1–element $\nu \in \mathcal{V}_\lambda\sqcup$ is in $\mathcal{TE}_1$, for $\lambda = 1, 2$. $\mathcal{I}(\mu) = \mu$ and $\mathcal{I}(\nu) = \nu$.

TE2 If $u \in \mathcal{T}_1\langle\mathrm{R}\rangle$, then $[\![u]\!] \in \mathcal{TE}_1\langle\mathrm{R}\rangle$. If $\tau \succeq \mathrm{R}$, $[\![u]\!](\tau) = [\![u(\tau)]\!]$. If $u \in \mathcal{T}_2\langle\mathrm{R}\rangle$, then $[\![u]\!]^- \in \mathcal{TE}_2\langle\mathrm{R}\rangle$. If $\tau \succeq \mathrm{R}$, $[\![u]\!]^-(\tau) = [\![u(\tau)]\!]^-$. (Recall, $[\![\cdot]\!]^-$ prevents the computation of the abstract instant -1).

TE3 If r is a relational expression over R, then $[\![r]\!]^- \in \mathcal{TE}_2\langle\emptyset\rangle$, and $[\![\mathcal{A}(r)]\!] \in \mathcal{TE}_1\langle\emptyset\rangle$. $\mathcal{I}([\![r]\!]^-) = +_{\tau\in\mathcal{I}(r)}[\![\tau]\!]^-$, and $\mathcal{I}([\![\mathcal{A}(r)]\!]) = +_{\tau\in\mathcal{I}(r)}[\![\mathcal{A}(\tau)]\!]$.

TE4 If $u_\lambda \in \mathcal{T}_\lambda\langle\mathrm{R}\rangle$ with A as its post, $u'_\lambda \in \mathcal{T}_\lambda\langle\mathrm{S}\rangle$ with B as its post, and A and B are θ-comparable, then $[\![u_\lambda \theta u'_\lambda]\!]^- \in \mathcal{TE}_\lambda\langle\mathrm{RS}\rangle$. If $\tau \succeq \mathrm{RS}$, then $[\![u_\lambda \theta u'_\lambda]\!]^-(\tau) = [\![u_\lambda(\tau)\theta u'_\lambda(\tau)]\!]^-$.

TE5 Suppose $t \in \mathcal{TE}_\lambda\langle\mathrm{R}\rangle$ and $t' \in \mathcal{TE}_\lambda\langle\mathrm{S}\rangle$. Then, $t+t'$, $t*t'$, and $t-t' \in \mathcal{TE}_\lambda\langle\mathrm{RS}\rangle$, and $-t \in \mathcal{TE}_\lambda\langle\mathrm{R}\rangle$, for $\lambda = 1, 2$. If $\lambda = 1$, then $fi(t)$ and $li(t) \in \mathcal{TE}_1\langle\mathrm{R}\rangle$. If $\tau \succeq \mathrm{RS}$, then $(t + t')(\tau) = t(\tau) + t'(\tau)$, $(t * t')(\tau) = t(\tau) * t'(\tau)$, and $(t - t')(\tau)$

$= t(\tau) - t'(\tau)$; and, if $\tau \succeq \mathrm{R}$, $-t(\tau) = \sqcup - t(\tau)$, $fi(t)(\tau) = fi(t(\tau))$, and $li(t)(\tau)$ $= li(t(\tau))$.

Example 7.2: Let us again consider the *personnel* database for a $\Box$–user. Suppose e is the temporal expression $[\![\mathsf{SALARY} \neq \mathsf{25K}]\!] + [\![\mathsf{DEPT} = \mathsf{Toys}]\!]$. According to our conventions, 25K stands for the 2-assignment $\langle \mathsf{NOW} \times [0, \infty) \mapsto \mathsf{25K} \rangle$ to the SALARY attribute. Similar remarks apply to the constant assignment "Toys". Suppose τ is John's tuple in the *emp* relation, $t = \mathsf{NOW}$ is the query anchoring instant, and $\tau' = \lceil\lceil \tau^t$. Then, $\mathsf{SALARY}(\tau')$ evaluates to the 1–assignment

$$\langle \{\mathsf{NOW}, -1\} \times ([11,49] \mapsto \mathsf{15K}, [50,54] \mapsto \mathsf{20K}, [55,60] \mapsto \mathsf{25K}) \rangle.$$

Now, $[\![\mathsf{SALARY} \neq \mathsf{25K}]\!](\tau')$ evaluates to

$$\begin{aligned} & \{\mathsf{NOW}, -1\} \times ([11,49] + [50,54]) \\ = \quad & \{\mathsf{NOW}, -1\} \times [11,54]. \end{aligned}$$

Similarly, $[\![\mathsf{DEPT} = \mathsf{Toys}]\!](\tau') = \{\mathsf{NOW}, -1\} \times [11,44]$. Thus, the expression $e(\tau) = \{\mathsf{NOW}, -1\} \times [11,54]$. The transaction timestamp always has the same value {NOW, -1} and can be omitted for the historical user. With this omission the result becomes the same as the result of $[\![\mathcal{A}(\mathsf{SALARY}) \neq \mathcal{A}(\mathsf{25K})]\!] + [\![\mathcal{A}(\mathsf{DEPT}) = \mathcal{A}(\mathsf{Toys})]\!]$. Thus, the concept of an anchor is redundant for the historical user.

□

7.1.3 Boolean expressions

Boolean expressions yield the truth values TRUE or FALSE when a tuple is substituted in them. The set $\mathcal{B}$ of all Boolean expressions is recursively defined as follows.

BE1 If $t \in \mathcal{TE}_\lambda\langle R\rangle$ and $t' \in \mathcal{TE}_\lambda\langle S\rangle$, for $\lambda = 1, 2$, then $t = t'$ and $t \subseteq t' \in \mathcal{B}\langle RS\rangle$. If $\tau \succeq RS$, $(t = t')(\tau)$ is TRUE if and only if $t(\tau) = t'(\tau)$; $(t \subseteq t')(\tau)$ is TRUE if and only if $t(\tau) \subseteq t'(\tau)$.

BE2 If $u \in \mathcal{T}_\lambda\langle R\rangle$ and $u' \in \mathcal{T}_\lambda\langle S\rangle$, then $u \subseteq u' \in \mathcal{B}\langle RS\rangle$. If $\tau \succeq RS$, then $(u \subseteq u')(\tau)$ is TRUE if and only if $u(\tau) \subseteq u'(\tau)$.

BE3 If $u \in \mathcal{T}_\lambda\langle R\rangle$ and $u' \in \mathcal{T}_\lambda\langle S\rangle$, such that the posts of u and u' are θ-comparable attributes, then $u\theta_A u'$ and $u\theta_S u' \in \mathcal{B}\langle RS\rangle$. If $\tau \succeq RS$, then $(u\theta_A u')(\tau)$ is TRUE if and only if $u(\tau)\theta_A u'(\tau)$ holds, and $(u\theta_S u')(\tau)$ is TRUE if and only if $u(\tau)\theta_A u'(\tau)$ holds. We use $u\theta u'$ as an abbreviation for the Boolean expression $u\theta_A u'$; in particular, $u = u'$ is an abbreviation for $u =_A u'$.

BE4 If $f \in \mathcal{B}\langle R\rangle$ and $g \in \mathcal{B}\langle S\rangle$, then $\neg f \in \mathcal{B}\langle R\rangle$, and $f \vee g$ and $f \wedge g \in \mathcal{B}\langle RS\rangle$. If $\tau \succeq R$, $\neg f(\tau)$ is TRUE if and only if $f(\tau)$ is FALSE; $(f \vee g)(\tau)$ is TRUE if and only if either $f(\tau)$ or $g(\tau)$ is TRUE, and $(f \wedge g)(\tau)$ is TRUE if and only if both $f(\tau)$ and $g(\tau)$ are TRUE.

BE5 If $r \in \mathcal{R}$ then $r = \emptyset \in \mathcal{B}\langle\emptyset\rangle$. $\mathcal{I}(r = \emptyset)$ is TRUE if and only if $\mathcal{I}(r)$ is $\emptyset$.

7.1.4 Relational expressions

Relational expressions are syntactic counterparts of 2-relations. Informally speaking, a relational expression is deemed valid if its interpretation is a "legal" 2–relation with a well-defined key. The set of all relational expressions $\mathcal{R}$ and their keys is defined as follows.

RE1 Every relation r $\in \mathcal{D}$ is in $\mathcal{R}$. We define $\mathcal{I}(r) = (\lceil\lceil r^t)\restriction'\sqcup - \{\emptyset\}$. This definition of $\mathcal{I}(r)$ restricts a user's view of $\mathcal{D}$ to $\mathcal{D}\restriction'\sqcup$, and extraneous information is

discarded.

RE2 If $r(\mathrm{R}) \in \mathcal{R}$ and I_{K} is a weak identity operator over R, then $I_{\mathrm{K}}(r) \in \mathcal{R}$ and K is its key. If $\mathcal{I}(r)$ is a 2–relation with K as its weak key, then we define $\mathcal{I}(I_{\mathrm{K}}(r)) = I_{\mathrm{K}}(\mathcal{I}(r))$; otherwise, it is undefined.

RE3 If $r(\mathrm{R})$ and $s(\mathrm{R}) \in \mathcal{R}$ and $\mathrm{K} \subseteq \mathrm{R}$ is their key, then $r \cup s \in \mathcal{R}$. If K is a weak key of the set $r \cup s$, then $\mathcal{I}(r \cup s)$ is defined to be $\{\tau \cup \tau' : \tau \in \mathcal{I}(r) \wedge \tau' \in \mathcal{I}(s) \wedge \tau \equiv_{\mathrm{K}} \tau'\} \cup \{\tau \in \mathcal{I}(r) : \neg\exists\tau' \in \mathcal{I}(s)(\tau \equiv_{\mathrm{K}} \tau')\} \cup \{\tau' \in \mathcal{I}(s) : \neg\exists\tau \in \mathcal{I}(r)(\tau \equiv_{\mathrm{K}} \tau')\}$, and K is its key; otherwise, $\mathcal{I}(r \cup s)$ is defined to be $\{\tau \cup \tau' : \tau \in \mathcal{I}(r) \wedge \tau' \in \mathcal{I}(s) \wedge \tau \equiv_{\mathrm{R}} \tau'\} \cup \{\tau \in \mathcal{I}(r) : \neg\exists\tau' \in \mathcal{I}(s)(\tau \equiv_{\mathrm{R}} \tau')\} \cup \{\tau' \in \mathcal{I}(s) : \neg\exists\tau \in \mathcal{I}(r)(\tau \equiv_{\mathrm{R}} \tau')\}$, and R is its key.

RE4 If $r(\mathrm{R})$ and $s(\mathrm{R}) \in \mathcal{R}$ and $\mathrm{K} \subseteq \mathrm{R}$ is their key, then $r - s \in \mathcal{R}$ and K is its key. If τ and τ' are 2–tuples such that $\tau \equiv_{\mathrm{K}} \tau'$, then define $\tau - \tau'$ as $\tau\restriction\mu$ where $\mu = \{t : \tau(\mathrm{A})\restriction t = \tau'(\mathrm{A})\restriction t \text{ for all } \mathrm{A} \in \mathrm{R}\}$. Then, we define $\mathcal{I}(r - s) = \{\tau - \tau' : \tau \in \mathcal{I}(r) \wedge \tau' \in \mathcal{I}(s) \wedge (\tau \equiv_{\mathrm{K}} \tau')\} \cup \{\tau \in \mathcal{I}(r) : \neg\exists\tau' \in \mathcal{I}(s)(\tau \equiv_{\mathrm{K}} \tau')\}$. The relational expression $r \cap s$ and its interpretation are defined in a similar manner.

RE5 If $r(\mathrm{R}) \in \mathcal{R}$ with K as its key, and X and K' are subsets of R, then $\Pi_{\mathrm{X}}(r; \mathrm{K}') \in \mathcal{R}$. If K' is a weak key for $s = \{\tau(\mathrm{X}) : \tau \in \mathcal{I}(r)\}$, then $\mathcal{I}(\Pi_{\mathrm{X}}(r; \mathrm{K}'))$ is defined to be $I_{\mathrm{K}'}(s)$; otherwise, it is defined to be $I_{\mathrm{X}}(s)$, and X is its key. In our syntax, we allow the K' in $\Pi_{\mathrm{X}}(r; \mathrm{K}')$ to be omitted, with the convention that if $\mathrm{K} \subseteq \mathrm{X}$ then K' defaults to K and otherwise it defaults to X.

RE6 If $r(\mathrm{R}) \in \mathcal{R}$ with K as its key, $f \in \mathcal{B}\langle\mathrm{S}\rangle$, $m \in \mathcal{TE}_2\langle\mathrm{S}\rangle$, and $n \in \mathcal{TE}_1\langle\mathrm{S}\rangle$, such that $\mathrm{S} \subseteq \mathrm{R}$, then $\sigma(r; f; m; n) \in \mathcal{R}$, and K is its key. We define $\mathcal{I}(\sigma(r; f; m; n))$

$= \{(\tau \restriction' m(\tau)) \restriction'_2 n(\tau) : \tau \in \mathcal{I}(r)$ and $f(\tau)$ holds$\}$ - $\{\emptyset\}$. This is an important operator in temporal databases. It allows a tuple to be selected on the basis of a Boolean condition (f), and then culled to the required time period (m) while also satisfying some criterion involving the anchor values for keys (n). Note that one or more of f, m, and n may be missing. If f is not specified it defaults to the Boolean value TRUE; if m is not specified it defaults to the value $\sqcup$; and if n is not specified it defaults to $\mho_2\sqcup$.

RE7 If $r_1(\mathrm{R}_1)$ and $r_2(\mathrm{R}_2) \in \mathcal{R}$, K_1 and K_2 are their respective keys, and $\mathrm{K} \subseteq \mathrm{R}_1\mathrm{R}_2$, then $r_1 \times r_2(\mathrm{K}) \in \mathcal{R}$. If K is the weak key for the set $s = \{(\tau_1 \circ \tau_2) \restriction' ([\![\tau_1]\!] * [\![\tau_2]\!]) : \tau_1 \in \mathcal{I}(r_1) \wedge \tau_2 \in \mathcal{I}(r_2)\}$, then $\mathcal{I}(r_1 \times r_2(\mathrm{K}))$ is defined to be $I_{\mathrm{K}}(s)$; otherwise, it is undefined — in this case a superset of K may work. In our syntax, we allow K in $r_1 \times r_2(\mathrm{K})$ to be omitted, and in that case it defaults to $\mathrm{K}_1\mathrm{K}_2$.

RE8 If $r(\mathrm{R}) \in \mathcal{R}$ with K as its key, $\mathrm{A} \in \mathrm{R}$ and $\mathrm{B} \notin \mathrm{R}$, then $\rho_{\mathrm{A} \leftarrow \mathrm{B}}(r) \in \mathcal{R}$. If $\mathrm{A} \notin$ K then K is the key for $\rho_{\mathrm{A} \leftarrow \mathrm{B}}(r)$; otherwise (K-A)UB is the key. $\mathcal{I}(\rho_{\mathrm{A} \leftarrow \mathrm{B}}(r))$ is the same as $\mathcal{I}(r)$, except that the attribute A is renamed to B.

7.2 Queries on 2–relations

An expression in the set $\mathcal{R} \cup \mathcal{TE}\langle\emptyset\rangle \cup \mathcal{B}\langle\emptyset\rangle$ is said to be *terminal*. A *query* is a terminal expression that has been defined with respect to an arbitrary, but fixed user domain $\sqcup$, and a fixed query anchoring point t. A user with user domain $\sqcup$ can only extract information from the $\prod(\mathcal{D} \restriction \sqcup)^t$.

Now we look at a number of examples of queries on 2–relations. All these queries refer to the personnel database of Figures 3.2 and 3.3.

7.2.1 The user domain ■

The ■–user has unrestricted access to the database. Besides using the algebraic constructs described in this chapter, such a user can also use the constructs for querying updates and errors. (These are described in Chapter 10.)

Example 7.3: Suppose ⊔ = ■, and t = NOW. This choice of t represents our belief that the database state is correct at NOW. Then the query "At what (2–dimensional) time was John's salary recorded as 15K, during the (real-world) time he was working in the Toys department?" is expressed as follows:

$$[\![\sigma(\text{emp}; \mathcal{A}(\text{NAME}) = \mathcal{A}(\text{John}); [\![\text{SALARY} = 15\text{K}]\!]^{-}; [\![\mathcal{A}(\text{DEPT}) = \mathcal{A}(\text{Toys})]\!])]\!]^{-}$$

□

In the above discussion we assumed that we are given a database $\mathcal{D}$ and a fixed user domain ⊔. If Q is a query, then its interpretation $\mathcal{I}(Q)$ depends both on $\mathcal{D}$ and ⊔. Therefore, it is more appropriate to write $\mathcal{I}_{⊔}(Q, \mathcal{D})$, instead of $\mathcal{I}(Q)$, when we have more than one user or database in mind. With this notation in place, we can state the following theorem:

Theorem 6: Let Q be a query and ⊔ a user domain. Then,

$$\mathcal{I}_{⊔}(Q, \mathcal{D}\restriction⊔) = \mathcal{I}_{■}(Q, \mathcal{D}\restriction⊔)$$

Proof: The proof follows from Theorem 3 in Section 6.1.

□

This theorem states that in order to answer a query for the ⊔–user, the system, ■, can evaluate the query directly from the stored database $\mathcal{D}$ in such a way that

the $\sqcup$–user gets the illusion that the query has been answered from the dedicated database $\mathcal{D}\restriction\sqcup$.

7.2.2 The user domain $\boxplus$

Suppose $\boxplus = t \times [0, \infty)$. If ξ is an anchored assignment, the transaction timestamp of ξ is $\{t, -1\}$. Also, the anchor $\mathcal{A}(\xi) = \pi_2^t(\xi)$ and the visible portion of the shielded part, $\pi_2^t(\mathcal{S}(\xi))$, are identical. Thus, the transaction time and the anchor of ξ serve no purpose for such a user and can be omitted in the interests of a more pleasant user interface. We observe that the difference between $[\![\xi]\!]$ and $[\![\xi]\!]^-$, and the difference between $[\![\xi_1 \theta \xi_2]\!]^-$ $[\![\xi_1 \theta \xi_2]\!]$ also disappear. The selection operator becomes simpler: such a user only needs $\sigma(r; f; m)$, and it is interpreted by the system as $\sigma(r; f'; m';)$ where f' and m' are obtained from f and m, respectively, by replacing every (1–dimensional) timestamp $\hat{t}$ by $(t, \hat{t})$

The following are examples of queries for a $\boxplus$–user. For such a user the version of history at transaction instant NOW is the correct one. All the examples refer to the historical version of the *personnel* database of Figures 6.1 and 6.2.

Example 7.4: List the starting salaries of the people currently employed.

$$\Pi_{\mathsf{NAME\ SALARY}}(\sigma(emp; \mathsf{NOW} \subseteq [\![\mathsf{DEPT}]\!]); fi([\![\mathsf{SALARY}]\!]))$$

Example 7.5: Give details about the employees who earned a salary greater than 24K while they worked in the Clothing or the Shoes department.

$$\sigma(emp;; [\![\mathsf{SALARY} > 24K]\!] * ([\![\mathsf{DEPT} = \mathsf{Shoes}]\!] + [\![\mathsf{DEPT} = \mathsf{Clothing}]\!]))$$

Example 7.6: Give details for all the employees in the Toys department during the time John was a manager in the company.

$$\sigma(emp;;[\![\sigma(management;;[\![\mathsf{MANAGER} = \mathrm{John}]\!])]\!] * [\![\mathsf{DEPT} = \mathrm{Toys}]\!])$$

7.2.3 The user domain ⊡

Suppose the user domain ⊔ is ⊡= $\{t\}\times\{t'\}$. Then, the timestamps for all objects in $\mathcal{D}$↾⊡ are the same. Additionally, all temporal expressions evaluate to either $\emptyset$ or $\{t\}\times\{t'\}$. As a result, the selection operator is seen as $\sigma(r;f)$ by such a user, and it is interpreted as $\sigma(r;f';;)$, where f' is obtained from f by attaching the timestamp $\{t\}\times\{t'\}$ to every non-timestamped data value in f. Such a user is not allowed to specify temporal expressions in any query. A special case of the ⊡–user is the □–user, discussed in the next subsection.

7.2.4 The user domain □

For the □–user, the timestamps for all data values in a relation are NOW×NOW. A query Q by a □–user can only refer to the time value NOW×NOW, and all the timestamps in the result relation, r, are NOW×NOW. For such a user the query interface is the same as that for a user in a classical relational database system. Before we prove this, we need a few auxiliary definitions.

We define the *extended snapshot algebra* to consist of the snapshot relational algebra, together with the Boolean operator *isempty*. If r is a snapshot relation then *isempty*(r) is TRUE if and only if $r = \emptyset$. A user of a classical snapshot database who has the extended snapshot algebra available is called an *extended snapshot user*.

Let r be a 2–relation in $\mathcal{D}$, over the scheme R. Then, define $S(r)$ to be the snapshot relation obtained by omitting the timestamps NOW×NOW from r↾NOW×NOW. We call S the *snapshot transformation function*. By extension, we can define $S(\mathcal{D}) = \{S(r) : r \in \mathcal{D}\}$. Also, for a Boolean f, define $S(f) = f$.

Suppose Q is a query for the $\Box$–user, resulting in relation r, and Q' is a query for the extended snapshot user, resulting in relation r'. We say that Q and Q' are *equivalent queries* if $S(r) = r'$. We say that the expressive power for the $\Box$–user is $\leq$ the expressive power for the extended snapshot user if for every query Q for the $\Box$–user, there is an equivalent query Q' for the extended snapshot user. The comparisons $\geq$ and $=$ are defined in a similar manner.

Lemma 7.1: The expressive power of the $\Box$–user $\leq$ the expressive power of the extended snapshot user.

Proof: Suppose the query Q for the $\Box$–user produces the answer a. Since the $\Box$–user is not allowed the use of temporal expressions, Q is a relational expression or a Boolean expression of the form $r = \emptyset$. In the former case we can construct Q' by substituting $S(r)$ for every occurrence of r in Q; in the latter case, Q' is simply *isempty*(r). Then, Q' produces the answer a' such that $S(a) = a'$.

Lemma 7.2: The expressive power of the $\Box$–user $\geq$ the expressive power of the extended snapshot user.

Proof: Since every query of an extended snapshot user is already a query for the $\Box$–user, the proof follows immediately.

Theorem 7: The expressive power of the $\Box$–user $=$ the expressive power of the

extended snapshot user.

7.3 Querying Extraneous Information

In our considerations, so far, we have excluded extraneous information. Recall, extraneous information arises when an anchor designates one version of history as the true one. Any information in the assignment (or tuple, or relation) that is about a real world instant for which the anchor has no information, is extraneous information. So far, for a relation r in the database, and for a given anchoring instant t, we have interpreted $\mathcal{I}(r)$ as $\lceil\rceil r^t$. However, at times extraneous information can be of value and we need to extend our algebra in order to query it.

Extraneous information enters the database because of two reasons: (i) We discover that a certain object τ does not exist in the real world. In that case $\lceil\rceil \tau^t = \emptyset$, and the whole tuple becomes extraneous. This causes a problem in providing interpretations for $r \cup s$ and $r - s$. (ii) We discover that, contrary to previous beliefs, an object is found not to exist for a certain period of time. In this case the object does have an identity through its key and there is no problem with $r \cup s$ and $r - s$. However, the extraneous information becomes an identity-less chunk once the key is changed. As a consequence, the following operations are affected by the inclusion of extraneous information:

- $I_{K'}(r)$.
- $\Pi_X(r, K')$ where K is the key of r and $K \neq K'$.
- $r \times s$.

The relational operators not affected are σ, ρ, and $\Pi_X(r)$, where X is contained in the key of r.

Suppose ξ_1 and ξ_2 are anchored assignments. Recall, $[\![\xi_1\theta\xi_2]\!]^-$ discards extraneous information, whereas $[\![\xi_1\theta\xi_2]\!]$ does not. Now we extend $\mathcal{T}$, $\mathcal{TE}$, $\mathcal{B}$, $\mathcal{R}$, and $\mathcal{E}$ to $\mathcal{T}^+$, $\mathcal{TE}^+$, $\mathcal{B}^+$, $\mathcal{R}^+$, and $\mathcal{E}^+$, respectively, to query extraneous information.

X1 Every expression in $\mathcal{T}$, $\mathcal{TE}$, $\mathcal{B}$, $\mathcal{R}$, and $\mathcal{E}$ is in $\mathcal{T}^+$, $\mathcal{TE}^+$, $\mathcal{B}^+$, $\mathcal{R}^+$, and $\mathcal{E}^+$, respectively.

X2 If $r(\mathrm{R})\in \mathcal{D}$ then $r^+ \in \mathcal{R}^+$. We define $\mathcal{I}(r^+) = r^t\natural'\sqcup$. Thus, the interpretation of r^+ does not reject any extraneous information.

X3 If $u \in \mathcal{T}_2^+\langle \mathrm{R}\rangle$ and $u' \in \mathcal{T}_2^+\langle \mathrm{S}\rangle$, then $[\![u\theta u']\!] \in \mathcal{TE}_2^+\langle \mathrm{RS}\rangle$. If $\tau \supseteq \mathrm{RS}$, then $[\![u\theta u']\!](\tau) = [\![u(\tau)\theta u'(\tau)]\!]$.

X4 If $r(\mathrm{R})\in \mathcal{R}^+$ with K as its key, and $\mathrm{K}\subseteq\mathrm{X}\subseteq\mathrm{R}$, then $\Pi_X(r;\mathrm{K}) \in \mathcal{R}^+$, and $\mathcal{I}(\Pi_X(r;\mathrm{K}))$ is defined to be $\{\tau(\mathrm{X})\colon \tau \in \mathcal{I}(r)\}$.

X5 If $r(\mathrm{R})\in \mathcal{R}^+$ with K as its key, $f \in \mathcal{B}^+\langle \mathrm{S}\rangle$, $m \in \mathcal{TE}_2^+\langle \mathrm{S}\rangle$, and $n \in \mathcal{TE}_1^+\langle \mathrm{S}\rangle$, such that $\mathrm{S}\subseteq\mathrm{R}$, then $\sigma(r; f; m; n) \in \mathcal{R}^+$, and K is its key. We define $\mathcal{I}(\sigma(r; f; m; n))$ $= \{(\tau\natural' m(\tau))\natural'_2 n(\tau) : \tau \in \mathcal{I}(r)$ and $f(\tau)$ holds$\} - \{\emptyset\}$.

X6 If $r(\mathrm{R}) \in \mathcal{R}^+$ with K as its key, $\mathrm{A}\in\mathrm{R}$ and $\mathrm{B}\notin \mathrm{R}$, then $\rho_{\mathrm{A}\leftarrow\mathrm{B}}(r) \in \mathcal{R}^+$. If $\mathrm{A} \notin \mathrm{K}$ then K is the key for $\rho_{\mathrm{A}\leftarrow\mathrm{B}}(r)$; otherwise $(\mathrm{K}-\mathrm{A})\cup\mathrm{B}$ is the key. $\mathcal{I}(\rho_{\mathrm{A}\leftarrow\mathrm{B}}(r))$ is the same as $\mathcal{I}(r)$, except that the attribute A is renamed to B.

Example 7.7: Give the entire information in the *emp* relation during the time John was a manager in some department.

$$\sigma(\mathrm{emp}^+;;[\![\sigma(\mathrm{management}^+;;[\![\mathcal{A}(\mathsf{MANAGER}) = \mathcal{A}(\mathrm{John})]\!];)]\!];)$$

8 UPDATES IN THE 2–DIMENSIONAL MODEL

In classical relational databases, we use the operations *insert*, *delete*, and *modify* to create, delete, and modify an object. This concept of updates is inadequate for our model, and we have to extend it. In this chapter we provide the formal semantics of updates in our model. This has to be done in order to properly introduce the update-querying capabilities of the 2–dimensional model. We also show how our model can "store" updates and how these can be recovered from the current state of the model.

In our model no information is ever deleted. The effect of a delete operation, in the classical sense, is handled by the use of the special symbol $\perp$, called *null*. The value $\perp$ in an assignment signifies that the object, or its relationship with other objects, does not exist at the 2–instant at which this value is taken. Note, we still require all tuples to be homogeneous — i.e., if one attribute has the value $\perp$ for a particular temporal interval, then so do all the other attributes. Thus a delete operation is effectively handled by setting all attribute values to $\perp$.

Before we give precise definitions for the update operations of our model, we need to look at some auxiliary definitions.

8.1 Extended Temporal Assignments

An *extended temporal assignment* ξ to an attribute A, with a λ–element μ as its *temporal domain* is a function from μ such that

1. For every $t \in \mu$, $\xi(t) \in \text{dom(A)} \cup \{\perp\}$.

2. The range of ξ is finite.

3. For each a in the range of ξ, $\{t : \xi(t) = a\}$ is a λ–element.

For an extended temporal assignment ξ, we define $[\![\xi]\!] = \{(t,t') : \xi(t,t') \neq \perp\}$. Thus, for extended temporal assignments $[\![\xi]\!]$ is *not* necessarily the same as its temporal domain.

For any 2–element μ, μ^{∞} denotes the 2–element $\mho_1(\mu) \times [0, \infty)$. Note that μ and μ^{∞} are the same in their transaction time dimensions.

If ξ is a λ–assignment, and ξ' an extended λ–assignment such that $\xi \subseteq \xi'$, $[\![\xi]\!] = [\![\xi']\!]$, and the temporal domain of ξ' is $\subseteq [\![\xi]\!]^{\infty}$, then ξ' is called the *null extension* of ξ. If the temporal domain of the null extension is $[\![\xi]\!]^{\infty}$, it is called the *total null extension* and is denoted by ξ^{∞}. An extended λ–assignment ξ is *total* if $[\![\xi]\!] = [\![\xi]\!]^{\infty}$.

If ξ_1 and ξ_2 are extended 2–assignments to θ–comparable attributes, we define $[\![\xi_1 \theta \xi_2]\!] = \{t : t \in [\![\xi_1]\!] * [\![\xi_2]\!] \ \wedge \xi_1(t) \theta \xi_2(t)\}$. Clearly, for temporal assignments ξ, ξ_1, and ξ_2, the contents of $[\![\xi]\!]$ and $[\![\xi_1 \theta \xi_2]\!]$ do not change under their respective null extensions. Thus, the symbol $\perp$ is used primarily for notational convenience; during updates (or queries) the user does not have to deal with $\perp$, nor is he required to input total λ–assignments. Furthermore, the user does not even have to provide the transaction time dimension — it is handled implicitly by the system.

The concept of extended tuples, extended relations, and extended databases can be defined in a natural manner, now. In the sequel, we shall drop the adjective "extended" in referring to them.

8.2 Update Operations

An update operation transforms a relation r from one state to another. An object (or, equivalently, a tuple) that is once inserted into a relation r is never deleted. Thus, we need update operations for creating new objects, and for modifying existing ones. We have made the assumption that the key of an object does not change in the real world. However, in practice it is not reasonable to expect that key values are always correct and that there are no errors in recording them. Therefore, we allow key values to be modified in order to correct the previous errors in recording them. This means that a tuple cannot just be identified with the values of its key attributes; we also need a time value for which the key attributes are being mentioned. In other words, the anchor value for key attributes is needed in order to uniquely identify an object.

An example of erroneous key values being modified can be found in the *emp* relation of Figure 3.2. In that relation, the identities of Leu and Inga were confused with one another, and at transaction time TT = 25 this mistake was corrected by interchanging their names. Note that such a change has to be made simultaneously in both the tuples; otherwise at some transaction time there would be two persons with the same name, thereby contradicting the assumption that NAME is the key for the *emp* 2-relation.

For the 2-dimensional temporal model, we define three update operations:

- *create* to create a single new object.

- *change* to modify the values and temporal domains of non-key attributes, and to modify the temporal domain of key attributes.
- *changekey* to modify the values of key attributes (simultaneously for several objects).

We do not exclude the possibility that the system may batch several update operations together, assigning them the same transaction time.

8.2.1 Some notational conventions

In order to give precise semantics for the update operations, we need some additional notation. Suppose ξ_1, ξ_2, $\cdots$, ξ_m are extended 1–assignments to distinct attributes A_1, A_2, $\cdots$, A_m, respectively, and μ is a 1–element. Then,

- $\mu \times \langle A_1 : \xi_1, A_2 : \xi_2, \cdots, A_m : \xi_m \rangle$ denotes the 2–tuple $\langle A_1 : \mu \times \xi_1, A_2 : \mu \times \xi_2, \cdots, A_m : \mu \times \xi_m \rangle$.
- $\langle A_1 : \xi_1, A_2 : \xi_2, \cdots, A_m : \xi_m \rangle \times \mu$ denotes the 2–tuple $\langle A_1 : \xi_1 \times \mu, A_2 : \xi_2 \times \mu, \cdots, A_m : \xi_m \times \mu \rangle$.
- If ξ and ξ' are 1–assignments to the same attribute, then $\xi \oplus \xi'$ denotes the 1–assignment $\xi\restriction([\![\xi]\!] - [\![\xi']\!]) \cup \xi'$.
- If ξ is a 2–assignment such that $\mathsf{NOW} \in \mathcal{V}_1[\![\xi]\!]$, then $\xi(\mathsf{NOW}/t)$ denotes the result of substituting every occurrence of NOW with the 1–instant t.

8.2.2 Update tags

Throughout this chapter we will make the assumption that we are given a fixed 2–relation r over the scheme $R = A_1 A_2 \cdots A_n$, with $K \subseteq R$ as its key, to which updates

are being made. Without loss of generality, we assume that $K = A_1 A_2 \cdots A_k$. Then, an *update tag* is a structure of the form $(A_1 : a_1,\ A_2 : a_2,\ \cdots,\ A_k : a_k;\ t)$, where $a_i \in dom(A_i)$, $1 \leq i \leq k$, and t a transaction instant in [0, NOW], is called an *update anchoring point.*

If τ is a tuple in r, and $t \in \mathcal{V}_1[\![\tau]\!]$, then we define $tag(\tau, t)$ to be the update tag $(A_1 : |\tau(A_1)\restriction_1 t|,\ A_2 : |\tau(A_2)\restriction_1 t|,\ \cdots,\ A_k : |\tau(A_k)\restriction_1 t|; t)$. It is clear that $tag(\tau, t)$ uniquely identifies the tuple τ in the relation r, i.e., for two tuples τ and τ' in r, if $t \in \mathcal{V}_1[\![\tau]\!]$, $t \in \mathcal{V}_1[\![\tau']\!]$ and $tag(\tau, t) = tag(\tau', t)$, then $\tau = \tau'$. For the sake of convenience, we assume that the *change* and *changekey* operations will only be performed for a tuple τ for which NOW $\in \mathcal{V}_1[\![\tau]\!]$. Thus we will always use NOW as the update anchoring point and not mention it in an update tag.

8.2.3 The create update operation

A *create* update operation is of the form

$$\textbf{create}(A_1 : \xi_1;\ A_2 : \xi_2;\ \cdots;\ A_n : \xi_n)\ \textbf{in}\ r; \tag{8.1}$$

where $\xi_1, \xi_2, \cdots, \xi_n$ are (extended) 1–assignments to $A_1, A_2, \cdots, A_n$, respectively, such that $[\![\xi_1]\!] = [\![\xi_2]\!] = \cdots = [\![\xi_n]\!]$. Suppose that a transaction time TT is given, and τ denotes the tuple $[TT, NOW] \times \langle A_1 : \xi_1,\ A_2 : \xi_2,\ \cdots,\ A_n : \xi_n \rangle$. If the set of tuples $r' = r \cup \{\tau\}$ is a 2–relation different from r with K as its key, the *create* operation is *accepted* by the system at transaction time TT and it transforms the relation r to r'; otherwise, the operation is *rejected.*

Example 8.1: The following *create* operation accepted at TT= 8 by the system, creates John's tuple in the *emp* relation of Figure 3.2.

create(NAME:⟨[11,∞) ↦ John⟩; SALARY:⟨[11,∞) ↦ 15K⟩; DEPT:⟨[11,∞) ↦ Toys⟩) **in** *emp*;

Example 8.2: Consider the following *create* operation for the *emp* relation of Figure 3.2.

create(NAME:⟨[81,90]+[101,110] ↦ Jean, [91,100]+[111,∞) ↦ ⊥⟩;
SALARY:⟨[81,85]+[101,110] ↦ 25K, [86,90] ↦ 30K, [91,100]+[111,∞) ↦ ⊥⟩;
DEPT:⟨[81,90]+[101,110] ↦ Toys, [91,100]+[111,∞) ↦ ⊥⟩)
in *emp*;

This *create* operation will be accepted by the system for the current state of the *emp* relation in Figure 3.2. In classical snapshot databases, one has to execute the following sequence of update operations to achieve the effect of this *one* update operation:

1. Insert Jean's tuple at TT = 81.
2. Modify SALARY at TT = 86.
3. Delete the tuple at TT = 91.
4. Again insert Jean's tuple at TT = 101, providing the same SALARY and DEPT information as before.
5. Again delete the tuple at TT = 111.

Also, since transaction time and real world time are coincident in snapshot databases, the above five steps would have to be executed at exactly the stated instants. Clearly, this is quite a limitation.

□

Lemma 8.1: Suppose the *create* operation C (8.1) transforms a relation r to r' at transaction time TT. Then, from r' alone we can calculate TT and C.

Proof: TT is the largest 1–instant satisfying $r'\restriction_1 t \neq r'\restriction_1 (t-1)$. Let τ be the (unique) tuple in r' such that $\tau\restriction_1(\mathsf{TT}-1)$ is null. Clearly, the create operation is of the form (8.1), where $\xi_i = \pi_2^{\mathsf{TT}}\tau(\mathrm{A}_i)$, $1 \leq i \leq n$.

□

8.2.4 The change update operation

A *change* operation is used to modify the values and temporal domains of non-key attributes and the temporal domains of key attributes. Suppose $(\mathrm{A}_1 : a_1,\ \mathrm{A}_2 : a_2,\ \cdots,\ \mathrm{A}_k : a_k)$ is an update tag, $\mathrm{B}_1,\ \mathrm{B}_2,\ \cdots,\ \mathrm{B}_l \subseteq \mathrm{R}$, and ξ_i is an extended 1–assignment to B_i, $1 \leq i \leq l$. Further, suppose that if B_i is a key attribute A_j, then $|\xi_i| = a_j$, $1 \leq i \leq l$, $1 \leq j \leq k$. Then a *change* operation is of the form:

$$\textbf{change}(\mathrm{A}_1 : a_1;\ \mathrm{A}_2 : a_2;\ \cdots,\ \mathrm{A}_k : a_k)\ \textbf{to}\ (\mathrm{B}_1 : \xi_1;\ \mathrm{B}_2 : \xi_2;\ \cdots,\ \mathrm{B}_l : \xi_l)\ \textbf{in}\ r; \tag{8.2}$$

This operation transforms a (unique) tuple τ having $(\mathrm{A}_1 : a_1,\ \mathrm{A}_2 : a_2,\ \cdots,\ \mathrm{A}_k : a_k)$ as its update tag, to a tuple τ' defined as follows. If TT is the transaction time at which the update takes place, then $\tau'(\mathrm{A}_1\mathrm{A}_2\cdots\mathrm{A}_k - \mathrm{B}_1\mathrm{B}_2\cdots\mathrm{B}_l)$ is the same as $\tau(\mathrm{A}_1\mathrm{A}_2\cdots\mathrm{A}_k - \mathrm{B}_1\mathrm{B}_2\cdots\mathrm{B}_l)$, and $\tau'(\mathrm{B}_i) = \tau(\mathrm{B}_i)(\mathsf{NOW}/\mathsf{TT}-1) \cup ([\mathsf{TT},\mathsf{NOW}]\times(\pi_2^{\mathsf{TT}-1}(\tau(\mathrm{B}_i)) \oplus \xi_i)$, for $1 \leq i \leq l$. If the set of tuples $r' = (r - \{\tau\}) \cup \{\tau'\}$ is a relation different from r, then we say that the *change* operation is

accepted by the system at transaction time TT, and that it transforms the relation r to r'; otherwise, it is *rejected.* Also, if there is no tuple in r with ($A_1 : a_1$, $A_2 : a_2$, $\cdots$, $A_k : a_k$) as its tag, the operation is rejected by the system.

Example 8.3: After the creation of John's tuple in the *emp* relation of Figure 3.2, the following *change* operations would result in its current state. For each *change* operation we also show the value of the transaction time TT when it is accepted by the system.

TT= 40: **change**(NAME: John) **to** (DEPT: ⟨[45,∞) ↦ Shoes⟩) **in** *emp*;

TT= 53: **change**(NAME: John)
to (SALARY: ⟨[50,∞) ↦ 20K⟩; DEPT: ⟨[50,∞) ↦ Shoes⟩) **in** *emp*;

TT= 55: **change**(NAME: John) **to** (SALARY: ⟨[55,∞) ↦ 25K) **in** *emp*;

TT= 60: **change**(NAME: John)
to (NAME: ⟨[61,∞) ↦ ⊥⟩; SALARY: ⟨[61,∞) ↦ ⊥⟩; DEPT: ⟨[61,∞) ↦ ⊥⟩)
in *emp*;

Note, in the last *change* operation, only the temporal domains are being changed. This operation is the equivalent of the delete operation in classical databases.

□

The *change* operation may sometimes be *partially redundant.* For instance, in the above sequence, the operation at TT = 53 is partially redundant, as the DEPT update can be omitted and "**change**(NAME: John) **to** (SALARY: ⟨[50,∞) ↦ 20K⟩) **in** *emp*;" would have the same effect.

Example 8.4: The following *change* operation will be rejected by the system as it does not alter the state of the *emp* relation:

TT= 90: **change**(NAME: John)

to (SALARY: ⟨[50,52] ↦ 20K⟩; DEPT: ⟨[50,52] ↦ Shoes⟩)

in *emp*;

Example 8.5: The following sequence of update operations shows a change being made in the value of an attribute, *before* the object actually comes into existence.

TT= 10: **create** (NAME: ⟨[30,∞) ↦ Jack⟩; SALARY: ⟨[30,∞) ↦ 22K⟩;

DEPT: ⟨[30,∞) ↦ Auto⟩)

in *emp*;

TT= 15: **change**(NAME: Jack)

to (NAME: ⟨[25,∞) ↦ Jack⟩; SALARY: ⟨[25,∞) ↦ 25K⟩;

DEPT: ⟨[25,∞) ↦ Auto⟩)

in *emp*;

Such an update sequence has interesting ramifications for the ◪–user. This user never gets to view the data until TT= 25, and is, therefore, unaware of the fact that "Jack's salary was changed from 22K to 25K."

8.2.5 Redundancy in the change operation

Consider a *change* operation C of the form (8.2), accepted by the system at transaction time TT, that causes a modification in a tuple τ in r. Then, C is said to be *non-redundant* if there does not exist an i, $1 \leq i \leq l$, and a 2–instant t such that $\pi_2^{TT-1}(\tau(B_i))\restriction t = \xi_i\restriction t$. Two change operations C and C′ are said to be *equivalent* with respect to r and a transaction time instant TT, written $C \equiv_{r,TT} C'$, or simply

C $\equiv$ C', if both C and C' transform r to the same state.

Lemma 8.2: Suppose the *change* operation C (8.2) transforms a relation r to r' at transaction time TT. Then from r' alone we can calculate TT, and a unique *change* operation C' such that C' is non-redundant and C $\equiv$ C'.

Proof: TT is the largest time instant t that satisfies $r'\restriction_1 t \neq r'\restriction_1 (t-1)$. Next, identify the unique tuple τ in r' such that $\pi_2^{TT}(\tau) \neq \pi_2^{TT-1}(\tau)$. In τ there is at least one attribute B such that $\tau(B)\restriction_1 TT \neq \tau(B)\restriction_1 (TT-1)$ — such an attribute is called a *critical* attribute. For the sake of convenience, denote $\tau(B)$ as ξ. The *change* operation C changes ξ only during $\delta = \{t' : \xi(TT,t') \neq \xi(TT-1,t')\}$. Since B is critical, $\delta \neq \emptyset$. Define $\xi_B = (\pi_2^{TT}(\xi))\restriction\delta$. Now, the *change* operation C' has the form (8.2) where (i) the tag used in C' is the same as the tag of $\tau\restriction(TT-1)$, and (ii) $B_1, B_2, \cdots, B_l$ are precisely the critical attributes, and each component of the form B:ξ is ξ_B as defined above.

□

The *change* operation C' in the lemma above is called *content*(C).

8.2.6 The changekey update operation

A *cyclic permutation* of a sequence $a_1, a_2, \cdots, a_m$ is another sequence having the form $a_j, a_{j+1}, \cdots, a_m, a_1, a_2, \cdots, a_{j-1}$, such that $1 \leq j \leq m$.

Suppose $UT_1, UT_2, \cdots, UT_m$ is a sequence of $m \geq 1$ update tags and the sequence $UT'_1, UT'_2, \cdots, UT'_m$ is its cyclic permutation. Then a *changekey* operation is of the form:

$$\textbf{changekey}\{UT_1\ \textbf{toUT}'_1;\ UT_2\ \textbf{toUT}'_2;\ \cdots;\ UT_m\ \textbf{toUT}'_m\}\ \textbf{in}\ r; \qquad (8.3)$$

Suppose a transaction time TT and tuples $\tau_1, \tau_2, \cdots, \tau_m \in r$ are given such that $tag(\tau_j) = \mathrm{UT}_j$, for $1 \leq j \leq m$. Define τ'_j(R–K) to be τ_j(R–K), and for each A∈K $\tau'_j(\mathrm{A}) = \tau_j(\mathrm{A})(\mathsf{NOW}/\ \mathsf{TT}-1) \cup [\mathsf{TT},\mathsf{NOW}]\times(\mathfrak{V}_2[\![\tau_j\restriction_1(\mathsf{TT}-1)]\!] \mapsto \mathrm{UT}'_j(\mathrm{A}))$. If $(r-\{\tau_1, \tau_2, \cdots, \tau_m\}) \cup \{\tau'_1, \tau'_2, \cdots, \tau'_m\}$ is a relation different from r with K as its key, we say that the *changekey* operation is *accepted* by the system at TT, and that it transforms r to r'; otherwise the operation is rejected. Also, if for some j, $1 \leq j \leq m$, there is no tuple $\tau \in r$ such that $tag(\tau) = \mathrm{UT}_j$, the *changekey* operation is rejected.

Example 8.6: The following update operations result in the current state of tuples for Inga and Leu in the *emp* 2-relation of Figure 3.2.

TT= 12:**create**(NAME: ⟨[71,∞) ↦ Leu⟩; SALARY: ⟨[71,∞) ↦ 25K⟩;
DEPT: ⟨[71,∞) ↦ Clothing⟩)
in *emp*;

TT= 14:**create**(NAME: ⟨[31,∞) ↦ Inga⟩; SALARY: ⟨[31,∞) ↦ 23K⟩;
DEPT: ⟨[31,∞) ↦ Toys⟩)
in *emp*;

TT= 25:**changekey**{(NAME: Inga) **to** (NAME: Leu);
(NAME: Leu) **to** (NAME: Inga) } **in** *emp*;

This *changekey* operation achieves the effect of correcting the error about the interchanged identities of Leu and Inga. The two tuples are changed simultaneously, as part of one atomic action.

Example 8.7: The following *changekey* operation corresponds to the discovery that the person hitherto known as Maria, is in fact Mary.

TT= 52: **changekey**{(NAME: Maria) **to** (NAME: Mary) } **in** *emp*;

Lemma 8.3: Suppose the *changekey* operation C (8.3) transform a relation r to r' at transaction time TT. Then, from r' alone we can calculate TT and C.

Proof: TT is the largest instant satisfying $r'\restriction_1 \mathsf{TT} \neq r'\restriction_1(\mathsf{TT}-1)$. In r' let $\{\tau_1, \tau_2, \cdots, \tau_n\}$ be the set of tuples such that, for some attribute A∈K, $|\tau_i(\mathrm{A})|$ is not a singleton and $|\tau_i(\mathrm{A})\restriction_1 \mathsf{TT}| \neq |\tau_i(\mathrm{A})\restriction_1(\mathsf{TT}-1)|$. Designate $tag(\tau_i, \mathsf{TT}-1)$ as UT_i and $tag(\tau_i, \mathsf{TT})$ as UT'_i for $1 \leq i \leq n$. Then C is of the form (8.3) with UT_i and UT'_i as described above.

□

We have already defined the concept *content* for a *change* operation. Now we extend it to the *create* and *changekey* operations by saying that for such an operation u, $content(u) = u$. Then, for any update operation u, we say that it is *minimal* if $content(u) = u$.

Now, the following follows from the three previous lemmas.

Theorem 8: Suppose an update operation u transforms a relation r to r' at transaction time TT. Then TT and $content(u)$ can be restored from r' alone.

8.3 Update Logs

In a conventional relational database system a log of updates is generally a non-relational entity. These information-rich logs are organized in an ad-hoc manner, without much structure, and without any means for querying them. In our 2–dimensional model, the update log is embedded within the database. Thus there is no need for

explicitly storing an update log. Additionally, the contents of this log can not only be queried using our query language, but also this log can be re-constructed from the current state of the database. Before we present our main results, we need a few definitions.

8.3.1 Some definitions

A *unit update transaction* is a pair $\langle u, \mathsf{TT} \rangle$, where u is an update operation and TT is a transaction time instant.

A *unit update log* is a finite sequence $\mathsf{UUL} = \langle u_1, \mathsf{TT}_1 \rangle, \langle u_2, \mathsf{TT}_2 \rangle, \cdots, \langle u_m, \mathsf{TT}_m \rangle$, of unit update transactions such that $i < j$ implies $\mathsf{TT}_i < \mathsf{TT}_j$, $1 \leq i, j \leq m$.

The unit update log $\mathsf{UUL} = \langle u_1, \mathsf{TT}_1 \rangle$, $\langle u_2, \mathsf{TT}_2 \rangle, \cdots$, $\langle u_m, \mathsf{TT}_m \rangle$, is *legal* if there exists a sequence $r_0 = \emptyset$, r_1, r_2, $\cdots$, r_m of 2-relations over the scheme R such that the update operation u_i transforms r_{i-1} into r_i at transaction time TT_i, for $1 \leq i \leq m$. In this case we say that r_m is the *outcome* of UUL.

Two unit update logs UUL^1 and UUL^2 are *equivalent*, written $\mathsf{UUL}^1 \equiv \mathsf{UUL}^2$, if their outcome is the same 2-relation.

8.3.2 Some results

Having given our preliminary definitions, we can now prove some results related to update logs.

Lemma 8.4: If $\mathsf{UUL}^1 = \langle u_1^1, \mathsf{TT}_1^1 \rangle$, $\langle u_2^1, \mathsf{TT}_2^1 \rangle$, $\cdots$, $\langle u_m^1, \mathsf{TT}_m^1 \rangle$, and $\mathsf{UUL}^2 = \langle u_1^2, \mathsf{TT}_1^2 \rangle$, $\langle u_2^2, \mathsf{TT}_2^2 \rangle$, $\cdots$, $\langle u_n^2, \mathsf{TT}_n^2 \rangle$, are equivalent unit update logs, then m=n,

$content(u_i^1) = content(u_i^2)$, and $TT_i^1 = TT_i^2$, for $1 \leq i \leq m$.

Proof: Let relations r_1 and r_2 be the respective outcomes of UUL^1 and UUL^2. Then, since $UUL^1 \equiv UUL^2$, it follows that $r_1 = r_2$. Therefore, for all t∈[0, NOW], $r_1 \restriction_1 t = r_2 \restriction_2 t$. This implies that in both r_1 and r_2 the *same* updates occurred at exactly the *same* transaction instants. Hence the proof.

□

Lemma 8.5: Suppose the unit update log $UUL = \langle u_1, TT_1\rangle, \langle u_2, TT_2\rangle, \cdots, \langle u_m, TT_m\rangle$, is legal, and $UUL' = \langle u_1', TT_1\rangle, \langle u_2', TT_2\rangle, \cdots, \langle u_m', TT_m\rangle$, is such that $u_i' = content(u_i)$, for $1 \leq i \leq m$. Then UUL' is equivalent to UUL.

□

Lemma 8.6: Let r be the outcome of the unit update log $UUL = \langle u_1, TT_1\rangle, \langle u_2, TT_2\rangle, \cdots, \langle u_m, TT_m\rangle$,, where each u_i, $1 \leq i \leq m$, is minimal. Then we can calculate the unit update log UUL from r alone.

Proof: $TT_1 = min(\{t : r \restriction_1 t \neq \emptyset\})$ and $\{TT_1, TT_2, \cdots, TT_m\} = \{TT_1\} \cup \{t > 0 : r \restriction_1 t \neq r \restriction_1 (t-1)\}$. The rest of the proof follows from Theorem 8.

□

8.3.3 Batched updates

So far we have assumed that a single update is executed at any given transaction instant. However, this does not have to be the case in general, and we can extend our framework as follows.

An *update transaction* is a pair $(\mathsf{U}, \mathsf{TT})$, such that U is a finite set of update operations and that no two update operations in U are for the same object.

Suppose r is a 2–relation over the scheme R, and $(\mathsf{U}, \mathsf{TT})$ is an update transaction, where $\mathsf{U} = \{u_1, u_2, \cdots, u_m\}$. We choose real numbers $\mathsf{TT}_1, \mathsf{TT}_2, \cdots, \mathsf{TT}_m$ in the interval $(\mathsf{TT}-1, \mathsf{TT}]$ such that $i < j$ implies $\mathsf{TT}_i < \mathsf{TT}_j$, for $1 \leq i, j \leq m$. We say that U is *accepted* by the system if there exists a sequence $r_0 = r, r_1, r_2, \cdots, r_m$ of 2–relations over the scheme R such that the update operation u_i transforms r_{i-1} into r_i at transaction time TT_i; otherwise U is *rejected.* If U is accepted by the system, replace every occurrence of TT_i by TT, $1 \leq i \leq m$, in r, to obtain the 2–relation r'. Then, r' is the *outcome* of U.

The concept of unit update logs and their legality, equivalence, outcome, and minimality can now be naturally extended to (non-unit) update logs. We now state the major result of this chapter.

Theorem 9: Suppose r is the outcome of the update log $\mathsf{UL} = \langle \mathsf{U}_1, \mathsf{TT}_1 \rangle, \langle \mathsf{U}_2, \mathsf{TT}_2 \rangle, \cdots, \langle \mathsf{U}_m, \mathsf{TT}_m \rangle$ such that U_i is minimal, for $1 \leq i \leq m$. Then, we can calculate the update log UL from r alone.

Proof: For each $\tau \in r$, calculate a unit update log $\mathsf{UUL}(\tau) = \langle u_1(\tau), \mathsf{TT}_1 \rangle, \langle u_2(\tau), \mathsf{TT}_2 \rangle, \cdots, \langle u_m(\tau), \mathsf{TT}_m \rangle$ whose outcome is the relation $\{\tau\}$. Define $\mathcal{TT}(\tau) = \{\mathsf{TT}_1, \mathsf{TT}_2, \cdots, \mathsf{TT}_m\}$, and $\mathcal{TT}(r) = \cup_{\tau \in r} \mathcal{TT}(\tau)$. We can think of $\mathsf{UUL}(\tau)$ as a set and define $\mathcal{UUL} = \cup_{\tau \in r} \mathsf{UUL}(\tau)$. For each transaction time $\mathsf{TT} \in \mathcal{TT}(r)$, consider $\mathcal{UUL}(\mathsf{TT}) = \{u : \langle u, \mathsf{TT} \rangle \in \mathcal{UUL}\}$. Let $\mathcal{UUL}'(\mathsf{TT})$ be obtained from $\mathcal{UUL}(\mathsf{TT})$ by collapsing the appropriate *changekey* operations together. Then $\mathsf{UL} = \{\langle \mathcal{UUL}'(\mathsf{TT}), \mathsf{TT} \rangle : \mathsf{TT} \in \mathcal{TT}(r)\}$ is the desired update log. (Strictly speaking, UL is a set, but it is easily

arranged as a sequence).

□

8.4 Remarks

In this chapter we have used NOW as the anchoring point, for ease of presentation. In general, any $t \in \mathcal{U}_1[\![\tau]\!]$ can be used. The need for a general update anchoring point is best justified by the following scenario. A tuple τ may be "deleted" from the database by setting its attribute values to $\perp$. If later, a need is felt to update some part of τ an anchoring point other than NOW is needed.

In classical databases the same object may be inserted and deleted several times over, without the knowledge at the logical level that it is the same object that is being updated. In our model, the *create* operation captures the first insertion of the object into the database. If the entire history of the object is known at the time of creation, this one operation is sufficient. Otherwise, *change* can be used to modify, delete, or un-delete (i.e., re-instate) the object. Further, the *changekey* operation allows errors made in recording key values to be rectified.

Our model does not place any simultaneity restrictions on the acceptance of a transaction and the time when a data value changes in the real world. An event can be recorded before it actually happens in the real world, or after it has occurred in the real world. Similarly, objects can be created, modified, or deleted either retro-actively or pro-actively.

Theorem 9 proves our claim that an update log is embedded in our model. This embedding provides users with the power to formulate queries about updates . This is the topic of the next chapter.

9 QUERYING FOR UPDATES AND ERRORS

In this chapter we introduce primitives for querying errors and updates in the database system. We incorporate these primitives into our relational algebra of Chapter 7. Thus we are able to write queries in the relational algebra that inform us of errors and updates

9.1 Primitives for Querying Updates

Suppose ξ is a 2-assignment to an attribute A in some relation in the database $\mathcal{D}$. Then we define the following sets of 2-instants:

$$\begin{aligned}
\delta_N(\xi) &= \{(t,t') : \xi(t,t') = \xi(t-1,t')\} \\
\delta_I(\xi) &= \{(t,t') : \xi(t,t') \neq \perp \ \wedge \xi(t-1,t') = \perp\} \\
\delta_D(\xi) &= \{(t,t') : \xi(t,t') = \perp \ \wedge \xi(t-1,t') \neq \perp\} \\
\delta_M(\xi) &= \{(t,t') : \xi(t,t') \neq \xi(t-1,t') \neq \perp\}
\end{aligned}$$

The sets δ_N, δ_I, δ_D, and δ_M are mutually disjoint temporal elements and their union is $[\![\xi]\!]$. These sets identify the 2-instants at which there are no-changes, insertions, deletions, and modifications, respectively, in the classical database sense. Updates in the sense of our model can be identified as follows.

1. $fi(\mho_1[\![\xi]\!])$ is the instant when ξ was created. $fi([\![\mathcal{A}(\xi)]\!])$ is the instant when ξ started existing in the real world, according to the correct version of reality.

2. If ξ is an assignment to a non-key attribute, then $(\delta_I(\xi) - fi(\mho_1[\![\xi]\!])) + \delta_D(\xi) + \delta_M(\xi)$ captures the 2–instants when a *change* operation was used to modify ξ.

3. If ξ is an assignment to a key attribute, then $\delta_M(\xi)$ captures the 2–instants when a *changekey* operation was used to modify ξ.

We can incorporate the above primitives for querying updates into our relational algebra of Section 7.3 by adding the following clause to the set of temporal expressions:

X7 For each attribute A, and for each $\delta \in \{\delta_N, \delta_I, \delta_D, \delta_M\}$, δ(A) is a temporal expression in $\mathcal{TE}^+\langle$A$\rangle$. If $\tau \succeq$A, then $\delta(\mathrm{A})(\tau) = \delta(\tau(\mathrm{A}))$.

Example 9.1: "When was John's salary changed?" This query is expressed as

$$\mho_1[\![\sigma(emp^+; \mathcal{A}(\mathsf{NAME}) = \mathcal{A}(\mathrm{John}); \delta_M(\mathsf{SALARY});)]\!].$$

Example 9.2: "Did Inga's identity ever need correction?"

$$\sigma(emp^+; \mathcal{A}(\mathsf{NAME}) = \mathcal{A}(\mathrm{Inga}); \delta_M(\mathsf{NAME});) \;=\; \emptyset.$$

9.1.1 Update queries and the user hierarchy

We define $\sqcup_\delta = \{(t, t') : (t-1, t') \in \sqcup\}$. Then, if ξ is an assignment to some attribute of a relation in $\mathcal{D}$, only the updates $\xi\restriction\sqcup_\delta$ are visible to $\sqcup$.

All updates are visible to the ■–user. The snapshot user, ⊡, has the capability to make any update, but cannot make any queries regarding updates. Intuitively,

such a user only sees one snapshot version of data and therefore has no concept of older or different values. Therefore he cannot query for the nature of updates.

A historical user, ⊡, sees one historical version of data. Therefore he too cannot compare his version of data with some other version. Thus, such a user cannot make queries for updates.

The ⧄–user represents the cumulative interaction of the ⧅–user over the transaction time axis. Thus, the ⧄–user sees the day by day evolution of the database, yet he cannot make any comparisons with any previously recorded values.

9.2 Primitives for Querying Errors

Suppose ξ is an assignment to an attribute A in some relation in the database $\mathcal{D}$, t' is a real world instant, and t_0 is an anchoring instant. According to our definitions, we consider the state of the database at the instant t_0 to be the correct version of reality. This means that we accept $\xi(t_0, t')$ as the correct information about the attribute A at the real world instant t'. Therefore, a comparison with $\xi(t, t')$ reveals the error made at instant t in recording reality for the real world instant t'. Such errors are classified as follows.

- No error if $\xi(t, t') = \xi(t_0, t')$.
- Missing information if $\xi(t, t') = \perp$ and $\xi(t_0, t') \neq \perp$.
- Extraneous information if $\xi(t, t') \neq \perp$ and $\xi(t_0, t') = \perp$.
- Incorrect information if $\xi(t, t') \neq \perp$ and $\xi(t_0, t') \neq \perp$ and they are not equal.

Corresponding to this classification of errors we define the following sets of 2–instants:

$$\varepsilon_N(\xi) = \{(t,t') : \xi(t,t') = \xi(t_0,t')\}$$
$$\varepsilon_M(\xi) = \{(t,t') : \xi(t,t') = \perp \wedge \xi(t_0,t') \neq \perp\}$$
$$\varepsilon_X(\xi) = \{(t,t') : \xi(t,t') \neq \perp \wedge \xi(t_0,t') = \perp\}$$
$$\varepsilon_I(\xi) = \{(t,t') : \xi(t,t') \neq \xi(t_0,t') \neq \perp\}$$

The sets ε_N, ε_M, ε_X, and ε_I are mutually disjoint and their union is $[\![\xi]\!]$. We can incorporate the above primitives for querying errors into our relational algebra of Section 7.3 by adding the following clause to the set of temporal expressions:

X8 For each attribute A, and for each $\varepsilon \in \{\varepsilon_N, \varepsilon_M, \varepsilon_X, \varepsilon_I\}$, ε(A) is a temporal expression in $\mathcal{TE}^+$(A). If $\tau \succeq$A, then $\varepsilon(\mathrm{A})(\tau) = \varepsilon(\tau(\mathrm{A}))$. Note, because of the homogeneity requirement we can omit mentioning the attribute name A in ε_M(A) and ε_X(A).

Example 9.3: "Give complete details about John when his salary was incorrectly recorded in the database." Using NOW as the anchoring point, this query is expressed as:

$$\sigma(emp^+; \mathcal{A}(\mathsf{NAME}) = \mathcal{A}(\mathrm{John}); \varepsilon_I(\mathsf{SALARY});)$$

Example 9.4: "Report extraneous information in the *emp* relation." This query may be expressed as:

$$\sigma(emp^+;;\varepsilon_X;)$$

Example 9.5: "Report the salary information for employees while their department information is not correct."

$$\Pi_{\mathsf{SALARY}}(\sigma(emp^+;;\neg\varepsilon_N(\mathsf{DEPT});))$$

Example 9.6: "Give information about managers in Toys, during the times there was an error in recording salaries for employees who were shown to be working in the Toys department." This query appears to be a join, but close examination reveals it to be a sequence of two selections.

$$\sigma(management^{+}; \mathcal{A}(\mathsf{DEPT}) = \mathcal{A}(\mathrm{Toys});$$
$$[\![\sigma(emp^{+};; \neg\varepsilon_{N}(\mathsf{SALARY}); [\![\mathsf{DEPT} = \mathrm{Toys}]\!];)]\!])$$

□

10 ZERO INFORMATION–LOSS

Conventional database systems are only capable of storing the current perception of reality, and the current relationship among objects. All old data values are excluded, and the only kind of historical information that can be obtained is through the transaction log. Even then, the transaction log has an ad hoc structure, and there is no theoretical model available for effectively querying these information-rich logs. Additionally, in such databases, an update operation is destructive – it not only destroys the environment in which it is executed, but also destroys the environment for queries. After an update is made, only the new database state is available, without even a clue to its past states.

A transaction in a database system is either an update, or a query. The activities in a database system consist, typically, of a sequence $\hat{T}$ of such transactions. In this chapter we formulate a *zero information–loss model*, in which no information is ever lost — not even the circumstantial information surrounding the transactions. A transaction in our model is recorded by imposing a logical structure on $\hat{T}$, in such a way that the precise effect of the transaction, as well as the transaction itself, can be determined at any time in the future. In addition, the logical structure imposed upon transactions gives rise to a simple, yet powerful, algebra.

10.1 Components of the Model

The zero information-loss model has three components. Of these, 2-relations constitute the basic model; shadow relations, and Q-rel constitute the other two components.

In a database system, the role of an update is to change the state of a database by modifying certain data values. At the same time, an update operation is associated with certain environmental parameters — the person authorizing the update, the reason for the update, the time of the update, etc. Such circumstantial data forms an important part of the information about an update, and is stored in certain relations called shadow relations. Corresponding to every 2-relation, r, we have a shadow relation, *shadow*(r), that is used for recording the circumstances surrounding the updates to r. The reason for separating r and *shadow*(r) is that their structures are very different; this separation allows us to harness the maximum potential of the relational approach for querying them effectively.

The Q-rel is a single relation used to store a log of all queries, together with information about their execution environments. With Q-rels, not only can we look at the details of (or, even re-execute) past queries, but also query queries, query queries on queries, ad infinitum. This extended capability has enormous potential in applications where it is important to monitor the flow of information from the database.

10.2 Transactions and the Zero Information–Loss Model

An update u to a relation r in our model can be decomposed into two parts: u^1, consisting of the new data value, and u^2, consisting of the circumstantial information surrounding the update u. The effects of u^1 and u^2 are reflected in r and *shadow*(r), respectively. This decomposition of the data and the circumstances of u is not lossy: the key of the updated tuple, and the transaction time of the update can be used to glue them back together. This property allows us to separate a relation r and its *shadow*, and therefore, we use separate structures for modeling them.

10.2.1 Definition of the zero information–loss model

Let R be a database scheme. Then a zero information–loss model over R is a triple $\langle\langle \mathcal{D}, shadow(\mathcal{D})\rangle,\ \mathrm{Q-rel}\rangle$, where

- $\mathcal{D}$ is the *basic model*, in which for every R$\in R$ there is a 2–relation r(R) $\in \mathcal{D}$.
- For every $r \in \mathcal{D}$ there is a relation *shadow*(r) $\in$ *shadow*($\mathcal{D}$).
- Q–rel is a single relation.

The structure $\langle \mathcal{D}, shadow(\mathcal{D})\rangle$ can be viewed as a logically structured log of all updates. This logical structure facilitates the querying of the environmental parameters associated with an update. Before giving the formal details of the update operations and the associated algebra, we present an example to provide motivation for the formalism.

10.2.2 A comprehensive example

Consider an empty instance of an *emp* 2–relation having the scheme NAME SALARY DEPT, with NAME as the key. Suppose that the shadow information corresponding to any update in the *emp* relation includes the transaction time (TT), the AUTHORIZER, the USER, and the REASON for the update. Now, consider the following sequence of transactions in the database:

T1 **create**(NAME: ⟨[11,∞) ↦ John⟩; SALARY: ⟨[11,∞) ↦ 15K⟩; DEPT: ⟨[11,∞) ↦ Toys⟩)

in *emp*;

TT = 8; AUTHORIZER = Don; USER = Mark; REASON = New Employee;

T2 **change**(NAME: John) **to** (DEPT: ⟨[11,44] ↦ Toys, [45,∞) ↦ Shoes⟩) **in** *emp*;

TT = 40; AUTHORIZER = Don; USER = Ryne; REASON = Reassignment;

T3 Q1: What is John's salary?

TT = 42; USER = Vance;

T4 **create**(NAME: ⟨[48,∞) ↦ Doug⟩; SALARY: ⟨[48,∞) ↦ 20K⟩; DEPT: ⟨[48,∞) ↦ Auto⟩)

in *emp*;

TT = 48; AUTHORIZER = Joe; USER = Rick; REASON = New Employee;

T5 **change**(NAME: John) **to** (SALARY: ⟨[11,49] ↦ 15K, [50,∞) ↦ 20K⟩) **in** *emp*;

TT = 53; AUTHORIZER = Don; USER = Damon; REASON = Promotion;

T6 Q1: What is John's salary?

TT = 54; USER = Andre;

T7 Q2: What is John's department?

TT = 55; USER = Mitch;

T8 Q3: Who made enquiries about John's salary?

TT = 56; USER = Don;

Corresponding to these transactions, the instance of our model $\langle\langle \mathcal{D}, shadow(\mathcal{D})\rangle$, Q–rel $\rangle$ is shown in Figure 10.1.

Note that the values of NAME in *emp* and NAME in *shadow(emp)* set up a logical correspondence between tuples of *emp* and *shadow(emp)*. By including the transaction time, this can be refined to a one-to-one correspondence between all updates to *emp* and the tuples in *shadow(emp)*. As a result, the transactions {T1, T2, T4, T5} can be completely restored from the current state of *emp* and *shadow*(*emp*), and {T3, T6, T7, T8} can be completely restored from *emp* and the Q–rel. Now, using TT, the transaction time, we can order {T1, T2, T4, T5} ∪ {T3, T6, T7, T8} to obtain the original sequence of transactions. Theorem 10 states that this is true in general, i.e., a transaction log can be restored from the current state of the zero information–loss model.

10.2.3 Shadow relations

Just as a 2–relation records the update made in the value of an attribute, the shadow relation records the environment of the update. Consequently, every 2–relation has a shadow relation associated with it, and for every update represented in the 2–relation, there is a corresponding tuple in the shadow relation.

Suppose $r \in \mathcal{D}$ is a 2–relation over R, with K⊆R as its key. Then the scheme of *shadow*(r), denoted *shadow*(R), necessarily contains {USER, TT} ∪ K. A USER is one

NAME		SALARY		DEPT	
[8,↝] × [11,∞)	John	[8,52] × [11,∞)	15K	[8,39] × [11,∞)	Toys
		[53,↝] × [11,49]	15K	[40,↝] × [11,44]	Toys
		[50,∞)	20K	[45,∞)	Shoes
[48,↝] × [48,∞)	Doug	[48,↝] × [48,∞)	20K	[48,↝] × [48,∞)	Auto

(a)The 2-relation

NAME	TT	AUTHORIZER	USER	REASON
John	8	Don	Mark	New Employee
John	40	Don	Ryne	Reassignment
Doug	48	Joe	Rick	New Employee
John	53	Don	Damon	Promotion

(b) The shadow relation

QUERY	TT	USER ID
Q1: John's SALARY	42	Vance
Q1: John's SALARY	54	Andre
Q2: John's DEPT	55	Mitch
Q3: USER ID of Q1	56	Don

(c) The Q–rel

Figure 10.1: The model ⟨⟨{*emp*}, {*shadow(emp)* }⟩, Q–rel⟩

who makes the update, and it can be a person, an algorithm, a machine, or any other decision making body. The transaction time TT is supplied by the system clock when an update becomes effective. The purpose of K is to establish a connection between an object in a 2–relation, and the corresponding tuple in its shadow. Other attributes in *shadow*(R) depend upon the nature of r. For example, in an employee shadow relation it might be relevant to record who authorized the salary update, whereas in a weather-record shadow relation it may make sense to record the space coordinates of the satellite reporting the information. Figure 10.1 shows a *shadow(emp)* relation, over NAME TT AUTHORIZER USER REASON.

10.2.4 Updates

Throughout this section we will assume that we are given a fixed but arbitrary 2–relation $r \in \mathcal{D}$ over the scheme R, with K$\subseteq$R as its key, and *shadow*(r) is a relation over *shadow*(R). Note that *shadow*(R) is required to contain {USER,TT} $\cup$ K. Unless otherwise noted, all references to values for attributes in the 2–relation refer to the correct values at the anchoring instant t = NOW. If $\tau \in r$ then *key*(τ) denotes the snapshot tuple $\hat{\tau}$ over K, such that for every A$\in$K, $\hat{\tau}(A) = |\tau(A)|^{NOW}|$.

In the zero information–loss model, an update operation transforms a 2–relation, and its corresponding shadow relation, from one state to another. The update operations on $\langle \mathcal{D}, shadow(\mathcal{D}) \rangle$ are as follows:

1. **create** τ **in** r
   with τ'.

2. **change** *key*(τ) **to** new–τ **in** r
   with τ'.

3. **changekey** $\{(key(\tau_1)$ **to** new_key_value$_1)$;
   $(key(\tau_2)$ **to** new_key_value$_2)$;
   $\vdots$
   $(key(\tau_n)$ **to** new_key_value$_n)\}$ **in** r
   with τ'.

where, τ' is the part of the update operation that has the shadow or circumstantial information about the update. τ' specifies the values of all attributes in *shadow*(R) – ({USER,TT}∪K); the USER, TT, and K information is automatically incorporated by the system.

An update operation u on $\langle \mathcal{D}, shadow(\mathcal{D}) \rangle$ can be partitioned into two parts: u^1 and u^2, which modify $\mathcal{D}$ and *shadow*($\mathcal{D}$), respectively. To make this partitioning more visible, and for the purpose of providing formal semantics, we first make *syntactic* modifications in our update operations so that an update operation u can be expressed as $\langle u^1, u^2 \rangle$:

1. ⟨**create** τ **in** r, **append** $\tau' \circ key(\tau)$ **to** *shadow*(r)⟩.

2. ⟨**change** $key(\tau)$ **to** new–τ **in** r, **append** $\tau' \circ key(\tau)$ **to** *shadow*(r)⟩.

3. ⟨**changekey** $\{(key(\tau_1)$ **to** new_key_value$_1)$;
   $(key(\tau_2)$ **to** new_key_value$_2)$;
   $\vdots$
   $(key(\tau_n)$ **to** new_key_value$_n)\}$ **in** r,

   append $\tau' \circ key(\tau_1)$ **to** *shadow*(r),
   append $\tau' \circ key(\tau_2)$ **to** *shadow*(r),

$\vdots$

append $\tau' \circ key(\tau_n)$ **to** *shadow*$(r)\rangle$.

If u is an update operator on $\langle \mathcal{D}, shadow(\mathcal{D})\rangle$, and u^1 and u^2 are such that $u = \langle u^1, u^2\rangle$, then u^1 and u^2 are update operations on r and *shadow*(r), respectively. Note that "**append** $\tau' \circ key(\tau)$ **to** *shadow*(r)" is the only update operation for the shadow relation *shadow*(r).

10.2.5 Semantics of update operations

In the previous subsection, we partitioned the three update operations of our model into separate operations on $\mathcal{D}$ and *shadow*$(\mathcal{D})$. In Chapter 8, we have already provided the semantics for the three update operations on $\mathcal{D}$; therefore, it only remains to give the semantics for the *append* operation on *shadow*$(\mathcal{D})$.

10.2.6 The append update operation

Suppose the current value of *shadow*(r) is denoted as s. Then, the update operation "**append** $\tau' \circ key(\tau)$ **to** *shadow*(r)" made by the user x, and accepted by the system at transaction time t, transforms the value of *shadow*(r) to $s \cup \{\tau' \circ key(\tau) \circ x \circ t\}$.

Lemma 10.1: Suppose s is a shadow relation over S. If s' is obtained from update u to s, by the user x, at transaction time t, then u, x, and t can be determined from s' alone.

Proof: Suppose s is *shadow*(r). Let τ be the unique tuple in s' such that $\tau(\mathsf{TT})$ is the largest. Then $u =$ "**append** $\tau(\mathsf{S} - (\{\mathsf{TT}, \mathsf{USER} \cup \mathsf{K}))\circ\tau(\mathsf{K})$ **to** r," $x = \tau(\mathsf{USER})$ and $t = \tau(\mathsf{TT})$.

□

Now the following lemma follows from Lemma 10.1 and Theorem 8.

Lemma 10.2: If $\langle r', shadow(r') \rangle$ are obtained after update u to $\langle r, shadow(r) \rangle$ by the user x at transaction time t, then $\langle r, shadow(r) \rangle$, u, x, and t can be restored from $\langle r', shadow(r') \rangle$ alone.

□

Definition: If u is the update $\langle u^1, u^2 \rangle$ such that u^1 creates, or modifies (*change* or *changekey*) τ, and u^2 creates tuple τ', then τ' is called the *shadow tuple* of τ.

10.2.7 Q–rel

Q–rel is essentially a log of all queries. A point of interest is that the same query may be made by different users at different times. For example, a compiled query program may be run at various times. The decision about how a query is to be stored — as a text string, or as a parse tree, or as some other semantic entity — is left to the choice of the implementer. We only require that there be some decision procedure for deciding whether or not two queries are the same. This decision procedure can be so naive as to label textually different queries as different, or so smart as to *try* and test equivalence of queries by some predetermined heuristic method.

We introduce a special attribute QUERY, and assume that dom(QUERY) consists of all queries. We define Q–*scheme* to be QUERY USER TT, and Q–rel to be a relation over Q–scheme. Thus, a tuple τ in Q–rel is of the form $\langle q, x, t \rangle$, where q is a query, x a user, and t a transaction time instant. The meaning of τ is that q was executed by the user x at transaction time t. Figure 10.1 shows an example of a Q–rel relation.

Suppose s represents the current state of Q–rel, the query q is asked by the user x, and the system answers it at transaction time TT. Then, the new state of Q–rel becomes $s \cup \{q \circ x \circ t\}$.

Lemma 10.3: Suppose the query q, by user x at transaction time t, transforms a Q–rel s to s'. Then s, q, x, and t can be restored from s' alone.

□

10.2.8 Information about a transaction

The information associated with a transaction is the data which is deemed relevant to the transaction under consideration. This definition places the burden of deciding relevance on the system designer. The system designer, then, has the responsibility of making *a priori* decisions about the kinds of circumstantial transaction data that needs to be stored and is to be recoverable. This is not an unreasonable requirement in light of the fact that databases systems only try to represent reality in a manner that is deemed adequate by the database scheme designers.

10.2.9 Transactions and transaction log

A transaction T in the zero information–loss model is of the form $\langle a, x, t \rangle$ where a is either an update or a query, x is a user, and t is a transaction time instant. If $\mathcal{Z}'$ is the state of a zero information–loss model after transaction T on the state $\mathcal{Z}$ of the model, then we write $\mathcal{Z}'$ as $T(\mathcal{Z})$.

Lemma 10.4: If T is a transaction on the state $\mathcal{Z}$ of the zero information–loss model,

then T and $\mathcal{Z}$ can be restored from $T(\mathcal{Z})$ alone.

Proof: Among all the tuples in all the shadow relations and the Q–rel, there is exactly one with the largest value in its TT attribute. Let τ be this tuple. If τ is in Q–rel, from Lemma 10.3 we know how to restore T and $\mathcal{Z}$. If τ is in some shadow relation *shadow*(r), then from Lemma 10.2 we know how to restore $\langle r, \mathit{shadow}(r)\rangle$, and consequently T and $\mathcal{Z}$.

□

Lemma 10.5: If T and T' are transactions such that $T(\mathcal{Z}) = T'(\mathcal{Z})$, then $T = T'$.

□

A *transaction log* is a sequence $\mathcal{TL} = \langle T_1, T_2, \cdots, T_n\rangle$ of transactions. Suppose $\emptyset$ denotes the initial empty state of the zero information–loss model. Then the *outcome* of the transaction log $\mathcal{TL}$ is the state $T_n(T_{n-1}(\cdots(T_1(\emptyset))\cdots))$.

Theorem 10: (The Zero–Loss Theorem) A transaction log $\mathcal{TL}$ can be restored from its outcome.

Proof: By repeated use of Lemma 10.4, we calculate $T_1, T_2, \cdots, T_n$ such that $\mathcal{Z} = T_1(\mathcal{Z}_1)$, $\mathcal{Z}_1 = T_2(\mathcal{Z}_2)$, $\cdots$, $\mathcal{Z}_{n-1} = T_n(\mathcal{Z}_n)$, and $\mathcal{Z}_n = \emptyset$. Clearly, $\mathcal{Z} = T_1(T_2(\cdots(T_n(\emptyset))\cdots))$. Therefore, $\mathcal{Z}$ is the outcome of the transaction log $\langle T_n, T_{n-1}, \cdots, T_1\rangle$. The fact that this is the transaction log which must have lead to $\mathcal{Z}$ follows from Lemma 10.5.

□

10.3 An Algebra for Zero Information–Loss Model

In this Section we extend our algebra of Chapter 7 to query the complete zero information–loss model.

10.3.1 Classical operators for shadow relations

Shadow relations are the classical 1nf relations, and we allow the classical relational operators on them.

10.3.2 Examples

Consider again the *emp* and *management* 2–relations of Figures 3.2 and 3.3, having NAME and DEPT as their respective keys. Suppose the schemes of *shadow(emp)* and *shadow(management)* are NAME TT AUTHORIZER USER REASON and DEPT TT AUTHORIZER USER REASON, respectively. The following queries are based on these relations. Whenever necessary, we implicitly rename an attribute A of a relation r to $r.A$.

Example 10.1: Give reasons for all the updates in *emp*.

$$\Pi_{\mathsf{REASONS}}(\mathit{shadow}(\mathit{emp}))$$

Example 10.2: Give reasons for all the updates in *emp* that were authorized by Harry.

$$\Pi_{\mathsf{REASONS}}(\sigma(\mathit{shadow}(\mathit{emp}); \mathsf{AUTHORIZER} = \text{Harry}))$$

Example 10.3: Who made changes in both *emp* and *management*?

$$\Pi_{\mathsf{USER}}(\mathit{shadow}(\mathit{emp})) \cap \Pi_{\mathsf{USER}}(\mathit{shadow}(\mathit{management}))$$

Example 10.4: . Who made the updates in *emp* between transaction time 10 and 20?

$$\Pi_{\mathsf{USER}}(\sigma(\mathit{shadow}(\mathit{emp}); \mathsf{TT} \geq 10 \wedge \mathsf{TT} \leq 20))$$

10.4 Navigation between $\mathcal{D}$ and *shadow*($\mathcal{D}$)

Suppose τ is a 2–tuple over R. If X$\subseteq$R, we define the 1–temporal element $\gamma(\tau(\mathrm{X}))$ $= \{t : \exists \mathrm{A} \in \mathrm{X},\ \exists t'(\tau(\mathrm{A})\restriction(t,t') \neq \tau(\mathrm{A})\restriction(t-1,t'))\}$. Clearly, $\gamma(\tau(\mathrm{X}))$ is the set of all transaction time instants when there is a change in $\tau(\mathrm{X})$. It is possible to break up the set $\gamma(\tau(\mathrm{X}))$ into two smaller sets:

- $\gamma_{\mathrm{I}}(\tau(\mathrm{X})) = \{t : \forall t' \in \mathcal{U}_2([\![\tau(\mathrm{X})]\!])(\tau(\mathrm{X})\restriction(t-1,t') = \perp)\}$.
- $\gamma_{\mathrm{C}}(\tau(\mathrm{X})) = \{t : \exists \mathrm{A} \in \mathrm{X}, \exists t' \in \mathcal{U}_2([\![\tau(\mathrm{X})]\!])(\tau(\mathrm{A})\restriction(t-1,t') \neq \tau(\mathrm{A})\restriction(t,t'))\}$.

Clearly, $\gamma(\tau(\mathrm{X}))$ is the disjoint union of $\gamma_{\mathrm{I}}(\tau(\mathrm{X}))$ and $\gamma_{\mathrm{C}}(\tau(\mathrm{X}))$. These two sets represent the transaction instants at which there are insertions and changes, respectively, for the tuple τ. If X is the set of key attributes then $\gamma_{\mathrm{C}}(\tau(\mathrm{X}))$ represents the instants at which the *changekey* operation is executed for τ.

Proposition: Suppose $r \in \mathcal{D}$ is a relation over R, with K$\subseteq$R as its key, $\tau \in r$ and $\tau' \in \mathit{shadow}(r)$ such that $\tau'(\mathsf{TT}) = t$. Then the following conditions are equivalent:

1. τ' is a shadow tuple of τ.
2. $\forall \mathrm{A} \in \mathrm{K}\ (|\tau(\mathrm{A})\restriction t)| = \tau'(\mathrm{A})$.

Also, if τ' is a shadow tuple of τ, then $\tau'(\mathsf{TT}) \in \gamma(\tau(\mathrm{R}))$.

Clearly, it would be useful to be able to express these conditions in our relational algebra. To make this possible, we now allow the following constructs.

1. The temporal expression $\gamma(X)$, where $X \subseteq R$. This expression is the syntactic counterpart of the set of those instants at which an attribute $A \in X$ was updated. Similarly, we have $\gamma_I(\tau(X))$ and $\gamma_C(\tau(X))$.

2. $|A|$, where $A \in R$, to represent the range of the temporal assignment to attribute A.

3. Boolean expressions of the form $p \subseteq q$, where p and q are finite subsets. $\mathsf{TT} \subseteq q$ is a special case, often written as $\mathsf{TT} \in q$.

The queries in the algebra for shadow relations can use temporal expressions involving relations in $\mathcal{D}$. This is illustrated in the following example.

Example 10.5: The temporal expression $[\![\sigma(\mathit{management}; \mathcal{A}(\mathsf{MANAGER}) = \mathcal{A}(\mathrm{Tom}); ;)]\!]$ retrieves the 2–element during which Tom is a manager in some department. Using this, the query "What were the reasons for updates in *emp* during the time Tom was a manager in some department?" may now be expressed as:

$$\Pi_{\mathsf{REASONS}}(\sigma(\mathit{shadow}(\mathit{emp}); \mathsf{TT} \subseteq [\![\sigma(\mathit{management}; \mathcal{A}(\mathsf{MANAGER}) = \mathcal{A}(\mathrm{Tom}); ;)]\!]))$$

10.4.1 Semijoins of relations in $\mathcal{D}$ and *shadow*($\mathcal{D}$)

Since the formats of a 2–relation and a shadow relation are quite different, we do not want to define their join. Instead we define semijoin operations [18], whose navigational nature is like a join, but instead of retrieving the concatenation of the tuples of the two operand relations, they only retrieve the tuples from *one* of the operand relations.

Suppose r is an (arbitrary) 2–relation over R with $K \subseteq R$ as its key, and s is an (arbitrary) shadow relation over S. For the sake of simplicity, we assume that $R \cap S$

$= \emptyset$; to accomplish this, whenever necessary we prefix *shadow* to an attribute of S. Suppose that f is a Boolean expression over RS. Then the *filter selection operators* are defined as follows:

1. $\ltimes_1(r;s;f) = \{\tau' \in s : \exists \tau \in r \wedge f(\tau \circ \tau')\}$

2. $\ltimes_2(r;s;f) = \{\tau \in r : \exists \tau' \in s \wedge f(\tau \circ \tau')\}$

Example 10.6: Give the complete employee record of everyone authorizing an update in *management*.

$$\ltimes_2(emp, shadow(management), |\mathsf{NAME}| = \mathsf{AUTHORIZER})$$

Example 10.7: Give the employees whose records were updated by Harry.

$$\ltimes_2(emp, shadow(emp), \mathsf{USER} = \mathrm{Harry} \wedge |\mathsf{NAME}| = shadow.\mathsf{NAME})$$

10.4.2 Filter operations

The special cases of the above semijoin operators when the operands r and s are subsets of a relation in $\mathcal{D}$ and *shadow*($\mathcal{D}$), respectively, and the Boolean expression $f(\tau \circ \tau')$ includes the conjunct $\mathsf{TT} \in \gamma(\mathrm{R})$, where R is the scheme of r are of considerable interest to us. Note that $\mathsf{TT} \in \gamma(\mathrm{X})$ is meant to express that "τ' is a shadow tuple of r which modified at least one attribute in X in τ." To ensure that we start with *whole* tuples, we have required r and s to be subsets of relations in $\mathcal{D}$ and *shadow*($\mathcal{D}$), respectively.

Formally, suppose $r(\mathrm{R}) \in \mathcal{D}$, $\mathrm{K} \subseteq \mathrm{R}$ is the key of R, f is a Boolean condition involving the attributes from R and *shadow*(R), and e_1 and e_2 are expressions which

evaluate to subsets of r and *shadow*(r), respectively. Then we introduce F_1 and F_2, called the *filter operations*, as follows.

1. $F_1(e_1, e_2, f) = \{\tau \in e_2 : \exists \tau' \in e_1(\mathsf{TT} \in \gamma(\mathrm{R}) \wedge f(\tau \circ \tau'))\}$.

2. $F_2(e_1, e_2, f) = \{\tau \in e_1 : \exists \tau' \in e_2(\mathsf{TT} \in \gamma(\mathrm{R}) \wedge f(\tau \circ \tau'))\}$.

Thus, F_1 returns a 1–nf relation, whereas F_2 returns a 2–relation.

Example 10.8: Which tuples in *emp* were updated by Harry?

$$F_2(emp, shadow(emp), \mathsf{USER} = \mathrm{Harry})$$

Example 10.9: Name the people who made changes in Tom's salary during [10,20].

$$\Pi_{\mathsf{USER}}(F_1(\sigma(emp; \mathsf{NAME} = \mathrm{Tom} \wedge \mathsf{TT} \in \gamma_C(\mathsf{SALARY}))))$$

10.5 Operators for Q–Rel

Recall, a tuple in Q–rel is of the form $\langle q, x, t\rangle$, and it says that the query q was executed by the user x at transaction time t. If τ is the tuple $\langle q, x, t\rangle$, then it is natural to identify it with the relation retrieved when q was executed at transaction time t. (Note that from x, we can decode the user class). We denote this relation as $[\tau]$. We define [Q–rel] = $\{[\tau] : \tau \in$ Q–rel$\}$. In this way we arrive at the database $Z = \mathcal{D} \cup$ *shadow*($\mathcal{D}$) $\cup$ Q–rel $\cup$ [Q–rel]. Z can be partitioned into two parts: Z^1 consisting of the classical 1-nf relations, and Z^2 consisting of 2–relations. On Z^1 we allow the classical relational operators, and on Z^2 we allow the algebra described in Chapter 7. Additionally, we can use the operators for navigation between $\mathcal{D}$ and

shadow($\mathcal{D}$). Clearly, the [Q–rel] part allows a query to be treated as a relation, which can be queried. Thus, we can query queries, query queries on queries, etc.

Example 10.10: Name the people who executed Q1.

$$\Pi_{\mathsf{USER}}(\sigma(\mathsf{Q}-\mathrm{rel}, \mathsf{QUERY} = \mathrm{Q1}));$$

Example 10.11: What difference was observed in execution of Q1 at TT = 3 and 10 by Harry?

$$[\mathrm{Q1}, \mathrm{Harry}, 10] - [\mathrm{Q1}, \mathrm{Harry}, 3]$$

Example 10.12: If Q is the query "What are all the naval bases?" executed by Vance at transaction time t, then $\dot{\sigma}$([Q,Vance,t]; LOCATION = Europe) can be used as an answer to the question "What naval base locations in Europe were revealed to Vance by the query Q?" and $\Pi_{\mathsf{COMMANDER}}$([Q,Vance,t]) can be used to answer the question "What COMMANDER information was released to Vance?"

□

The above examples demonstrate the utility of querying the Q–rel. This has great application in areas where it is necessary to monitor the accesses being made to the database. Such query capability allows the database administrator to analyze the information-flow patterns.

11 QL2: A QUERY LANGUAGE FOR THE MODEL

In the 2–relational model, the relations contain a complete historical record of the data. A user can access the entire history, a particular version of history, or just an instantaneous "snapshot" of data. The user domain determines what information can be made available to a user. Thus, the user domain controls the information that can be provided to a user. Consequently, every database query takes the user domain into account.

The 2–relational model allows a user to query for errors made in modeling data by comparing versions of history with some known "true" version of history. This true version of history is determined by the *anchoring point*, which is a transaction time instant. The history of data, as recorded at the anchoring point, is stored as a special entity that cannot be accessed explicitly, but which can be used to provide the 2–assignment with its true identity. This identity is used in θ–comparisons.

From the above discussion it is clear that every query has a user domain and an anchoring point associated with it. These two entities are part of the environment of the query and are set outside the body of the query.

Within the body of a query, a user can ask for the following kinds of entities:

1. **Relations:**

   Example: Give the employee record for everyone in the Toys department.

2. **2–Temporal Elements:**

   Example: What was the period of Tom's employment?

3. **1–Temporal Elements:** These can be from either the transaction time or the real world time dimension.

   Example: For what transaction time period did we think that Tom was working in Shoes?

   Example: For what real world time period was Tom's salary 30K?

4. **Boolean Answers:**

   Example: Are there any employees in the Toys department?

Thus, a query in QL2 should be able to return the above mentioned types of data objects. For ease of description, we divide queries in QL2 into two classes:

1. Basic queries.

2. Boolean queries.

11.1 Preliminary Definitions

Before giving the syntax and semantics of the query constructs in QL2, we need to look at a few definitions.

An *aliaser* of a relation r is an expression of the form $r \rightarrow s$. The purpose of such an expression is to create a tuple variable s which ranges over the relation r. The aliaser $r \rightarrow r$ is simply written as r, and aliasers $r \rightarrow s_1$, $r \rightarrow s_2$, $\cdots$, $r \rightarrow s_m$ can be written as $r \rightarrow s_1, s_2, \cdots, s_m$.

A *from-list* is a list of aliasers of the form $(r_1 \rightarrow s_1;\ r_2 \rightarrow s_2;\ \cdots;\ r_n \rightarrow s_n)$, such that $i \neq j$ implies $s_i \neq s_j$, $1 \leq i,j \leq n$.

Example 11.1: (*emp* $\rightarrow$ e_1, e_2; *management*) is a from-list. It creates variables e_1 and e_2 that range over the *emp* relation, and the variable *management* to range over the *management* relation. An equivalent statement in SQL would be "*emp* e_1, *emp* e_2, *management*."

□

A *renamer* ρ of a relation scheme R is a one-to-one function from R into the universal set of attributes. A renamer ρ of the scheme $A_1A_2\cdots A_n$ may be written as $(A_1 \rightarrow \rho(A_1),\ A_2 \rightarrow \rho(A_2),\ \cdots,\ A_n \rightarrow \rho(A_n))$, where $A \rightarrow \rho(A)$ may simply be written as A if $\rho(A) = A$. If ρ is a renamer of R and τ is a tuple over $S \supseteq R$, then $\rho(\tau)$ denotes the tuple τ' over $\rho(R)$, such that for all $A \in R$ $\tau'(\rho(A)) = \tau(A)$.

A *result operator*, $\mathcal{O}$, having a set r as its operand, is one of the following:

1. $\{\cdot\}$ — the identity function.
2. $[\![\cdot]\!]$.
3. $\mathcal{U}_\lambda$, for $\lambda = 1, 2$.

The *homogeneous product* of 2-relations $r_1,\ r_2,\ \cdots,\ r_n$ is defined as follows:

$$r_1 \times r_2 \times \cdots \times r_n = \{(\tau_1 \circ \tau_2 \circ \cdots \circ \tau_n)\restriction([\![\tau_1]\!] * [\![\tau_2]\!] * \cdots * [\![\tau_n]\!]) : \tau_i \in r_i \text{ for } 1 \leq i \leq n\}$$

This operation basically takes a cross product of the tuples in the relation and then homogenizes this product by restricting it to the intersection of the temporal domains of the tuples.

11.2 Basic Queries

A basic query in QL2 consists of the following *select statement*:

SELECT	<target object>
FROM	<from-list>
RESTRICTED TO	<2-temporal expression>
WHERE	<Boolean expression>
DURING	<1-temporal expression>
INTO	<result variable>

Informally, when such a query is executed, a relation is computed by first choosing the tuples from the homogeneous cross-product of the relations mentioned in <from-list> that satisfy the <Boolean expression>, and then by temporally restricting these tuples. These tuples are restricted so that their temporal domains conform to the <2-temporal expression>, and their anchor values satisfy the <1-temporal expression>. Any resulting null-tuples are discarded. Then, a result operator is applied to this relation to produce the final answer.

It should be noted that in a query, only the **SELECT** and **FROM** clauses are necessary. The other parts may be omitted. We will discuss this in detail in Section 11.5.

In what follows, we may assume without loss of generality that the attributes of relations named in the <from-list> of a query are all different. Let ATTR be this set of all the attributes.

11.2.1 Syntactic classes in a basic query

In this section we look at the details of the syntactic classes mentioned in the description of a basic query, above.

1. **Target Object:** The target object specifies the kind of result that is to be returned by the query. The possible options are:

   $\{\rho\}$; **K,** where ρ is a renamer and K is a set of attributes contained in ATTR. With this specification, the query returns the result relation after restructuring it so that K forms a key for it. The specification of K is optional, and when left unspecified, the entire scheme of the resulting relation is assumed to be the key. The braces $\{\}$ represent the identity result operator defined earlier, and are optional. A special case is when instead of ρ, the symbol $*$ is used; in the tradition of SQL, the symbol $*$ stands for the entire tuple.

   $\mathcal{O}(\rho)$, where $\mathcal{O}$ is either the $[\![\cdot]\!]$ or the $\mho_\lambda$ result operator, $\lambda = 1$ or 2. In this case the temporal domain of the result relation is returned.

2. **From–List:** In this class, any from–list can be specified. Inside the from–list any base relation or any computed relation can be mentioned.

3. **2–Temporal Expressions:** In this class any valid 2–temporal expression can be used. (Including, clauses X7 and X8 from Chapter 9).

4. **Boolean Conditions:** In this class any valid Boolean expression in $\mathcal{B}$, described in Chapter 7, can be used.

5. **1–Temporal Expressions:** This can be any expression from $\mathcal{TE}_1$, described in Chapter 7.

6. **Result Variables:** These are names for storing the result of a query. The type of the result variable must correspond to the type of object returned by the query — relations, 2–temporal elements, 1–temporal elements, etc.

11.2.2 Procedural semantics of a basic query

Now, let q be the basic QL2 query:

SELECT	$\mathcal{O}(\rho)$; K
FROM	$r_1 \rightarrow s_1, r_2 \rightarrow s_2, \cdots, r_k \rightarrow s_k$
RESTRICTED TO	μ
WHERE	f
DURING	ν
INTO	x

where $\mathcal{O}$ is a result operator, ρ is a renamer, $\mathrm{K} \subseteq \mathrm{ATTR}$, $r_1, r_2, \cdots, r_k$ are 2–relations, μ is a 2–temporal expression, ν is a 1–temporal expression, f is a Boolean formula involving attributes from ATTR, and x is a variable of the appropriate result type.

Then, the result of evaluating q is the following:

$$x := I_{\mathrm{K}}\Big(\bigcup_{\tau \in (r_1 \times r_2 \times \cdots \times r_k) \wedge f(\tau)} \mathcal{O}(\rho((\tau \restriction \mu) \restriction_2 \nu)) \Big)$$

Example 11.2: What was the period of Tom's employment?

SELECT	$[\![*]\!]$
FROM	*emp*
WHERE	NAME = Tom

Example 11.3: For what transaction time period did we think that Tom was working in the Shoes department?

SELECT	$\mathcal{U}_1(*)$
FROM	*emp*
RESTRICTED TO	[[DEPT = Shoes]]
WHERE	NAME = Tom

Example 11.4: For what real world time period was Tom's salary 30K?

SELECT	$\mathcal{U}_2(*)$
FROM	*emp*
RESTRICTED TO	[[SALARY = 30K]]
WHERE	NAME = Tom

Example 11.5: At what (2-dimensional) time was John's salary recorded as 15K in the database during the real world time he was (really) working in Toys?

SELECT	[[*]]
FROM	*emp*
RESTRICTED TO	[[SALARY = 15K]]
WHERE	NAME = John
DURING	[[DEPT = Toys]]

Example 11.6: Get details for employees who earned a salary greater than 24K while they were in the Clothing or the Shoes department.

SELECT	*
FROM	*emp*
RESTRICTED TO	[[SALARY> 24K]]*
	([[DEPT =Shoes]]+ [[DEPT = Clothing]])

Example 11.7: List the name and starting salary of those currently employed.

SELECT	{NAME SALARY}; NAME
FROM	*emp*
RESTRICTED TO	*fi*[[SALARY]]
WHERE	(NOW⊆ [[DEPT]])

11.3 Boolean Queries

Such queries are used for yes/no answers from the database. The basic form of a Boolean query is:

<Boolean Query Expression> **INTO** <Boolean variable>

where, a Boolean query expression is defined as follows.

1. If r is a basic QL2 query then $r = \emptyset$ is a Boolean query expression.
2. If r_1 and r_2 are basic QL2 queries, then $r_1 \subseteq r_2$ is a Boolean query expression.
3. If q is a Boolean query expression then so is (q).
4. If q_1 and q_2 are Boolean query expressions, then so are q_1 **AND** q_2 and q_1 **OR** q_2.

The **INTO** clause is optional, and the result variable in that clause is of Boolean type.

Example 11.8: Are there any employees in the Toys department?

(SELECT *
FROM *emp*
WHERE DEPT = Toys) = ∅

11.4 Other Query Forms

Besides the queries described above, QL2 allows queries involving union, intersection, and difference of expressions.

The syntax of such queries is:

(q1 QUERYOP q2)
INTO <result variable>

where,

1. QUERYOP is one of **UNION**, **INTERSECTION** , or **MINUS**, and
2. q1 and q2 are QUERYOP compatible relational expressions.

Again, the **INTO** clause is optional, and the result variable in it must be of relational type.

11.5 The QL2 Hierarchy

The QL2 hierarchy is a parallel of the user hierarchy described in Chapter 6. Recall, the user hierarchy is determined by the user domains in such a way that users at a higher level in the hierarchy can access everything that users at lower levels can. The description of QL2 that we have provided above, is for the 2–user. Users at lower levels are allowed more limited access and, therefore, do not have all of the features described above. For example, the snapshot user cannot use the **RESTRICTED TO** and the **DURING** clauses of QL2. As a result, for the snapshot user, QL2 appears to be the same as SQL.

11.5.1 The historical user

The historical user, □, can only view 1–dimensional temporal information. Therefore, for the □–user, the **DURING** clause is redundant. Additionally, the □–user specifies 1–temporal expressions in the **RESTRICTED TO** clause of a basic query.

For a given □–user there is a fixed transaction time, t_0, at which the user views the data. For such a user, the temporal universe is 1–dimensional, spanning only real world time. Every temporal reference by such a user is one dimensional and refers to one dimensional time. However, when the system responds to such a user's query, it converts every 1–dimensional temporal reference μ to $\{t_0\} \times \mu$. It uses this 2–dimensional temporal expression *privately*, and returns answers with only 1–dimensional temporal references — the □–user gets the impression of a dedicated historical database.

11.5.2 The snapshot user

The snapshot user, ⊡, looks at only one snapshot of data corresponding to a 2–instant $\{t_1\} \times \{t_2\}$. Such a user is not allowed to make any temporal references in his queries, and thus is not allowed to use the **RESTRICTED TO** and the **DURING** clauses. Analogous to the case for the ⊞–user, when the system responds to such a user's query, it attaches the 2–dimensional time stamp $\{t_1\} \times \{t_2\}$ to every static (non-temporal) reference. Again, it uses this 2–dimensional temporal expression *privately*, and returns answers with only static references — the ⊡–user gets the impression of a dedicated snapshot database.

11.6 Queries on Shadow Relations & Q–Rel

QL2 has constructs for querying shadow relations and Q–rel. Therefore, it is also an adequate query language for the zero information-loss model.

11.6.1 Queries on shadow relations

Queries on shadow relations can be classified into the following two categories, based on the kinds of relations they use as operands:

Pure Queries: These are queries which only use shadow relations, and produce results which are static relations. For such queries, QL2 uses the syntax and constructs of SQL [11], and we shall not discuss these here.

Mixed Queries: These are queries which use shadow relations, as well as 2–relations from $\mathcal{D}$. These queries correspond to the filter operations defined in Chapter

10. Before giving the QL2 constructs for such queries, we define the following new Boolean constructs in QL2.

- **ATTRIBUTE = X:** Let $X = A_1 A_2 \cdots A_k$ be a set of attributes. Then the construct ATTRIBUTE = X is used to express the Boolean condition $TT \subseteq \gamma(X)$ where $\gamma(X)$ is the set of those transaction instants when an update was recorded for some attribute $A_i \in X$, $1 \leq i \leq k$.
- **|A| = domain constant:** Such an expression is used for comparing a domain constant with the data value associated with a 2–assignment to an attribute A. Such expressions are particularly useful for navigation between 2–relations and static relations.
- **|A| = B:** Such an expression is used for comparing the data value associated with a 2–assignment, to an attribute A, with the value associated with the static attribute B. Such expressions are particularly useful for navigation between 2–relations and static relations.

Now we can describe the constructs for the filter operations F_1 and F_2:

1. **Construct for $F_1(r, s, f)$:**

S_SELECT	ρ
FROM	r
AND	s
WHERE	f
INTO	x

Here, ρ is a renamer, r is a 2–relation over R, s is a shadow relation over R′, f is a Boolean condition involving attributes from R and R′, and x is a result variable of

the appropriate type.

The effect of executing a query such as above is:

$$x := \{\rho(\tau) : \tau \in F_1(r, s, f)\}$$

2. Construct for $F_2(r, s, f)$:

2_SELECT	ρ;K
FROM	r
AND	s
WHERE	f
INTO	x

Here, ρ is a renamer, r is a 2–relation over R, K$\subseteq$R, s is a shadow relation over R$'$, f is a Boolean condition involving attributes from R and R$'$, and x is a result variable of the appropriate type.

The effect of executing a query such as above is:

$$x := I_K\{(\rho(\tau) : \tau \in F_2(r, s, f)\}$$

Example 11.9: Give reasons for all the updates in emp.

SELECT	REASON
FROM	*shadow(emp)*

Example 11.10: Give reasons for all the updates that were authorized by Harry.

SELECT	REASON
FROM	*shadow(emp)*
WHERE	AUTHORIZER = Harry

Example 11.11: Who made changes in both *emp* and *management*?

```
(SELECT        USER
FROM           shadow(emp))
INTERSECTION
(SELECT        USER
FROM           shadow(management))
```

Example 11.12: Who updated employee salaries between transaction time 10 and 20?

```
SELECT         USER
FROM           shadow(emp)
WHERE          (ATTRIBUTE = SALARY)
               AND (TT≤ 10) AND (TT≤ 20)
```

Example 11.13: What were the reasons for updates in *emp* during the time Tom was a manager in some department?

```
SELECT         REASON
FROM           shadow(emp)
WHERE          TT ⊆ (SELECT      υ1(*)
                     FROM        management
                     RESTRICTED TO
                                 [[MANAGER = Tom]])
```

Example 11.14: Give the complete employee record of everybody who authorized an update in *management*.

2_SELECT	*
FROM	*emp*
AND	*shadow(management)*
WHERE	|NAME| = AUTHORIZER

Example 11.15: Give the employees whose records were updated by Harry.

2_SELECT	*
FROM	*emp*
AND	*shadow(emp)*
WHERE	(USER = Harry) AND
	|NAME| = *shadow(emp)*.NAME

11.6.2 Queries on Q–rel

The zero information-loss model includes the special relation Q–rel, which is basically a relational log of all queries. To query Q–rel, QL2 provides a simple construct:

Q_SELECT	ρ
WHERE	f
INTO	x

Here, ρ is a renamer, f is a Boolean expression involving attributes from Q-SCHEME, and x is a result variable.

Notice, the **Q-SELECT** construct does not have a **FROM** clause. This is because of the fact that there is a unique Q–rel for every zero information-loss model.

QL2 includes another construct that allows a user to re-execute a past query. This is accomplished by specifying the query-id and the transaction time at which the

state of the zero information-loss model is to be used in the computation. The result of such an operation is a relation corresponding to the execution of the query at the specified transaction time instant. The QL2 construct for this operation is:

EVAL	Q
AT	t
INTO	x

Here, Q is the query-id, t is the transaction time of its execution, and x is a result variable.

11.6.3 Examples

Recall that the Q-SCHEME includes the attributes QUERY, USER, and TT that specify the query, its user, and the time at which it was executed.

Example 11.16: Name the people who executed the query Q1.

Q_SELECT	USER
WHERE	QUERY = Q1

Example 11.17: What difference was observed in the execution of the query Q1 at TT=3 and TT=10? Store the answer in RESULT.

((EVAL	Q1
AT	10)
MINUS	
(EVAL	Q1
AT	3))

INTO RESULT

11.7 QL2 Constructs for Performing Updates

QL2 provides constructs for updating 2–relations in $\mathcal{D}$. As every update has some shadow information associated with it, these constructs also allow the user to specify some of this circumstantial shadow information (part of the shadow information is determined automatically).

11.7.1 The CREATE operation

This operation is used for introducing new tuples in a 2–relation. As the transaction time dimension can be determined automatically, the user of this operation has to specify 1-assignments for the introduced tuple. (For a snapshot user, both the time dimensions are determined automatically.) The QL2 construct for this operation is as follows:

CREATE τ
IN r
WITH τ'

Here, τ is the tuple being created in the 2–relation r, and τ' is the shadow tuple that is to be appended to *shadow*(r).

Example 11.18: The following QL2 statement creates the tuple for John in the *emp* relation.

CREATE (NAME: $\langle[11,\infty) \mapsto$ John$\rangle$;

	SALARY: ⟨[11,∞) ↦ 20K⟩;
	DEPT: ⟨[11,∞) ↦ Toys⟩)
IN	*emp*
WITH	(AUTHORIZER = Don;
	REASON = New Employee)

Note, only 1–assignments have been specified for the tuple being created. The system determines the transaction time of this operation and converts each timestamp into a 2–dimensional timestamp. Also, shadow information such as USER, TT, and $key(\tau)$ is automatically computed and stored in the shadow tuple.

11.7.2 The CHANGE operation

This operation is used for changing the assignments to non-key attributes and the temporal domain of assignments to key attributes of a tuple. The tuple to be changed is identified by specifying its key. The QL2 construct is as follows:

CHANGE	$key(\tau)$
IN	r
TO	new-τ
WITH	τ'

Here, the tuple identified by $key(\tau)$, in r, is changed to new-τ, with the corresponding shadow information τ' being stored in $shadow(r)$.

Example 11.19: Consider the following change in DEPT for John's tuple in the *emp* 2–relation:

CHANGE	NAME: John
IN	*emp*
TO	(DEPT: ⟨[11,44] ↦ Toys, [45,∞) ↦ Shoes⟩)
WITH	(AUTHORIZER = Don;
	REASON = Re-assignment)

Again, the USER, TT, and $key(\tau)$ are determined and stored with the other shadow information.

11.7.3 The CHANGEKEY operation

This operation allows the keys of tuples to be changed (simultaneously). It is used to model the real-world event of a mistake being uncovered in the previous recording of the key value for an object. The QL2 construct is as follows.

CHANGEKEY	$\{(\tau_1$ **TO** $\tau_1');$
	$(\tau_2$ **TO** $\tau_2');$
	$\vdots$
	$(\tau_n$ **TO** $\tau_n')\}$
IN	r
WITH	τ'

Here, the τ_i and τ_i' are key values and τ' is the shadow information that is stored with the shadow tuples associated with each of the τ_i's.

Example 11.20: The names (key value) of two people, Leu and Inga, have been mistakenly interchanged in the *emp* relation. This mistake may be corrected as follows:

CHANGEKEY	{(NAME:Leu **TO** NAME:Inga);
	(NAME:Inga **TO** NAME:Leu)}
IN	*emp*
WITH	(AUTHORIZER = Don;
	REASON = Mistaken Identity)

Again, the USER, TT, and $key(\tau)$ attributes are automatically determined and stored in the shadow tuples of both the tuples being updated here.

12 CONCLUSIONS

In this thesis we have proposed a temporal relational database model with 2–dimensional timestamps. This model is a consistent extension of the classical relational model and includes a robust concept for keys. The concept of keys plays a central part in the definition of the relational algebra for our model.

The inherent complexity of temporal data manifests itself in a number of ways in the 2–dimensional model. Since there are multiple versions of object histories, we get the concept of extraneous information. Such information does not lend itself to restructuring, and this dictates the form of the relational algebra for extraneous information. Anchors are used in our model to provide objects with consistent and unambiguous identities. For key attributes, anchors play a especially useful part during query formulation.

The concept of relation restructuring is central to our model. A change in key for a relation causes a relation to be restructured along the new key. As a result, parts of one temporal assignment combine with parts of others to form objects (tuples) with new identities. However, restructuring is controlled in the sense that transaction units are not allowed to be split.

Our model supports a user hierarchy with varying levels of privileges. Users at the lowest level in the hierarchy can only view that part of the 2–dimensional model that corresponds to the classical database model. Others, at higher levels, have higher

access privileges.

The relational algebra in the classical relational model has the property that expressions yield objects that can be used as operands for even bigger expressions. This important property is also present in the algebra for our model. Consequently, it is possible to write complex queries that use the results of other (sub-)queries as their operands. The query language QL2 provides constructs for all the clauses of the relational algebra. This language is SQL based and provides a friendly user interface.

In our work we have identified primitives for querying errors and updates. Naturally, this is only possible because the 2–dimensional model has the power to provide a formal framework for these concepts. The query language QL2 includes constructs that facilitate the formulation of such queries.

The zero information-loss model represents an important application of the 2–dimensional model. The 2–dimensional model, in conjunction with structures such as Q–rel and shadow relations, obviates the need for explicit transaction logs [3]. Not only that, the zero information-loss model has the capability to provide information about transactions. Thus, it forms a basis for building auditable and secure database systems.

12.1 Future Directions

This thesis has provided a theoretical model for temporal databases. Such a model has many practical applications. One such application — zero information-loss systems — has been formally defined here. Future work would study implementation strategies and performance issues for such models.

An important extension of the relational algebra includes the concept of aggre-

gates. For the 2–dimensional model, an open question concerns the nature of such operations and the semantics attached to their results. Such issues have to be resolved while making sure that the results are not objects of types that cannot be used as operands in bigger expressions.

In recent years, there has been a lot of emphasis on non-1NF relational databases. Bancilhon and Khoshafian [2] provides a calculus for complex objects that can be used to model non-1NF structures. A formulation of the 2–dimensional temporal relational model along these lines is an interesting possibility. The effectiveness of such an approach is a largely unexplored issue.

13 ACKNOWLEDGEMENTS

I am grateful to my thesis advisor, Dr. Shashi K. Gadia, for his help during the course of this work. Not only did Dr. Gadia introduce me to this problem, he constantly provided me with direction and guidance. During the last several months, Dr. Gadia was always readily available with advice and reassurance. He always had kind words of encouragement whenever I had the "Ph. D. blues." Dr. Gadia's assistance was never more than a phonecall away.

I would like to thank the other members of my Program of Study committee — Dr. R. D. Maddux, Dr. A. E. Oldehoeft, Dr. G. M. Prabhu, and Dr. A. Pohm — for their time and energy. Dr. Prabhu's comments on the first draft of my research proposal helped improve the quality of that effort considerably.

I am also grateful to the wacky dudes in the System Support Group — Chris, Dave (#1), Dave (#2), and Quent — for all their help with LaTeX matters. Also, thanks to Lakshman Mukkavilli for help with METAFONT. Mrs. Bishop and the ISU Thesis Office staff provided comments about the layout of the thesis.

On a personal level, it goes without saying that none of this would have ever been possible without the constant encouragement from my parents and family. Finally, my heartfelt thanks to Reena for so graciously putting her career on hold, and for just being there.

Gautam Bhargava.

14 BIBLIOGRAPHY

[1] Abadi, M., and Z. Manna. "Temporal Logic Programming." *Proceedings of 1987 IEEE Symposium on Logic Programming*, San Francisco, CA, August, 1987.

[2] Bancilhon, F., and S. Khoshafian. "A Calculus for Complex Objects." *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1986.

[3] Bhargava, G., and S. K. Gadia. "Achieving Zero Information-Loss in a Classical Database Environment." *Proceedings of the Fifteenth International Conference on Very Large Data Bases*, Amsterdam, The Netherlands, 1989.

[4] Bolour, A., T. L. Anderson, L. Dekeyser, and H. Wong. "The Role of Time in Information Processing: A Survey." *SIGART Newsletter* 80(1982):28–48.

[5] Bontempo, C. J. "Feature Analysis of Query-By-Example." *Relational Database Systems*. New York: Springer–Verlag, 1983.

[6] Chomicki, J., and T. Imielinski. "Temporal Deductive Databases and Infinite Objects." *Proceedings of the Seventh Annual ACM SIGACT-SIGMOD Symposium on the Principles of Database Systems*, Austin, TX, March, 1988.

[7] Clifford, J., and A. Tansel. "On an Algebra for Historical Relational Databases: Two Views." *Proceedings of ACM SIGMOD International Conference on Management of Data*, Austin, TX, May, 1985.

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

[9] Codd, E. F. "A Relational Model of Data for Large Shared Shared Data Bank." *Communications of the ACM* 13(1970):377–387.

[10] Copeland, G., and D. Maier. "Making Smalltalk a Database System." *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1984.

[11] Date, C. J. *An Introduction to Database Systems*, Reading, MA, Addison-Wesley Publishing Company, 1986.

[12] Gadia, S. K. "Weak Temporal Relations." *Proceedings of the Fifth Annual ACM SIGACT-SIGMOD Symposium on the Principles of Database Systems*, Los Angeles, CA, 1986.

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

[14] Gadia, S. K., and J. Vaishnav. "A Query Language for a Homogeneous Temporal Database." *Proceedings of the Fourth Annual ACM SIGACT-SIGMOD Symposium on the Principles of Database Systems*, 1985.

[15] Gadia, S. K., and C. Yeung. "A Generalized Model for a Relational Temporal Database." *Proceedings of ACM SIGMOD International Conference on Management of Data*, Chicago, IL, June, 1988.

[16] Gadia, S. K., and C. Yeung. "Inadequacy of Interval Timestamps in Temporal Databases." To appear in *Information Sciences.*

[17] Khoshafian, S. N., and G. Copeland. "Object Identity." *Proceedings of OOPSLA*, New York, 1986.

[18] Maier, D. *The Theory of Relational Databases.* Rockville, MD, Computer Science Press, 1983.

[19] McKenzie, E. "Bibliography: Temporal Databases." *ACM SIGMOD Record* 15(1986):40–52.

[20] McKenzie, E., and R. Snodgrass. *Supporting Valid Time: An Historical Algebra.* Technical Report TR87-008. Computer Science Department, University of North Carolina at Chapel Hill, August, 1987.

[21] Montague, R. "The Proper Treatment of Quantification in Ordinary English." *Approaches to Natural Language.* Ed. K. Hintikka. Dordrecht, West Germany: D. Reidel Publishing Company, 1973.

[22] Navathe, S., and R. Ahmed. *A Temporal Relational Model and Query Language.* Technical Report, Database Systems R & D, University of Florida, 1986.

[23] Palley, N. A., et al. *CLINFO User's Guide: Release One.* Technical Report R-1543-1-NIH, Rand Corporation, Santa Monica, CA, 1976.

[24] Segev, A., and A. Shoshani. "Logical Modelling of Temporal Data." *Proceedings of ACM SIGMOD International Conference on Management of Data*, San Francisco, CA, 1987.

[25] Shoshani, A., and K. Kawagoe. "Temporal Data Management." *Proceedings of the Twelfth International Conference on Very Large Data Bases*, Kyoto, Japan, 1986.

[26] Snodgrass, R. "Research Concerning Time in Databases: Project Summaries." *ACM SIGMOD Record* 15(1986):19–39.

[27] Snodgrass, R. "The Temporal Query Language TQUEL." *ACM Transactions on Database Systems* 12(1987):247–298.

[28] Snodgrass, R., and I. Ahn. "A Taxonomy of Time in Databases." *Proceedings of ACM SIGMOD International Conference on Management of Data*, Austin, TX, May, 1985.

[29] Snodgrass, R., S. Gomez, and E. McKenzie. *Aggregates in the Temporal Query Language TQuel.* TempIS Technical Report 16, Computer Science Department, University of North Carolina at Chapel Hill, July, 1987.

[30] Stam, R. B., and R. Snodgrass. "A Bibliography on Temporal Databases." *ACM SIGMOD Record* 15(1986):53–61.

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

[32] Tansel, A. U. "A Statistical Interface for Historical Relational Databases." *Proceedings of the Third IEEE International Conference on Data Engineering*, Los Angeles, CA, February, 1987.

[33] Tansel, A. U., and L. Garnett. "Nested Historical Relations." *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1989.

[34] Weiderhold, G., J. Fries, and S. Weyl. "Structured Organization of Clinical Data Bases." *Proceedings of the National Computer Conference, AFIPS*, 1975.