A SURVEY OF
TEMPORAL DATA MODELS

BY

STEVEN M. REIMER

A Thesis

Submitted to the Faculty of Graduate Studies

in Patrtial Fulfillment fo the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Computer Science

University of Manitoba

Winnipeg, Manitoba

ISBN 0-315-76759-6

A SURVEY OF TEMPORAL DATA MODELS

BY

STEVEN M. REIMER

A thesis submitted to the Faculty of Graduate Studies of
the University of Manitoba in partial fulfillment of the requirements
of the degree of

MASTER OF SCIENCE

TABLE OF CONTENTS

1. INTRODUCTION

1.1 INTRODUCTION

A database in the Codd's relational data model [13] has been defined by C. J. Date [15] as "a collection of time-varying, normalized relations of varying degrees." "Time-varying" means that the set of tuples in the relation vary over time. The data, therefore, has an implicit time dimension and history associated with it.

The set of tuples which comprise a relation in this data model are a snapshot of the data as it existed at some point in time. An update to a relation is "destructive" in the sense that current data is replaced with a new value and the relation maintains no record of the previous value. The only record of the past data is retained through backup copies of the data and transaction logs which record the sequence of updates to the data. The time "dimension" of the data has been lost.

The time component of data must be maintained though, to support some applications of the data. Decision support systems for example, are used by managment to analyze data to aid in business planning. Decisions are often based on retrospective and trend analysis of historic data. A corporate personnel database may be used to perform historic salary analysis or to examine an employee's job history to perform succession planning. Other database applications, such as financial record history and version histories in engineering design, require historic data be maintained through a "non-deletion policy". Many scientific and statistical databases maintain time dependent information in the form of the results of the physical experiments, measurements and simulations over the time domain.

With present relational database technology, access to past data is provided by recreating past states of the

database with backup copies and transaction logs or by incorporating the support for time related data into the application program itself. The first alternative, recreating past database data with backup copies and transaction logs, is not an acceptable solution since dynamic, ad hoc access to historic data is not possible without first recreating the database. The second alternative, incorporating time support into the application program, results in a more complicated database because of the additional time attributes. This database will have data redundancies, and perform less efficiently as more data is added. Within an organization, the application programs which maintain and access this data will do so in an *ad hoc* manner due to different approaches in manipulating the time attributes.

It is for these reasons that support for data that varies over time should be incorporated into the data model and the resulting database management system (DBMS). In this way, time-varying data will be handled in a uniform and consistent manner, thereby increasing the efficiency and maintainablitiy of the resulting applications and databases. When time-varying data support is included in the data model, the database schema is designed to model only the current occurrence and the DBMS automatically maintains history. As an example, consider the job history for an employee. In a DBMS which supports time-varying data, an employee database is designed as if the employee has only one job. Every time the job information for an employee is updated though, historic data is maintained automatically by the DBMS. Thus, many job occurrences are maintained for an employee without actually having to design this into the database.

1.2 DOMAIN OF INTEREST

The time related data of interest are sequences of data values occurring over the time domain and the operations which manipulate these sequences of data. An example of such data is the salary history for an employee. Initially, the salary may be $20,000 when the employee is hired (May 1, 1988). After the first

year, the salary increases by ten percent to $22,000 (May 1, 1989) and again after the second year it increases by a further ten percent to $24,200 (May 1, 1990). A sequence of data values over time will be referred to as temporal data. A data model provides a definition for the structure of objects used to represent entities being modeled and a collection of operations to manipulate the objects. A data model which provides support for temporal data will be referred to as a temporal data model.

Another type of time/data relationship which may be of interest is that of "relative temporal data" as proposed by Allen [6] and Chaudhuri [9]. Relative temporal data is that in which one data value or event preceeds or follows another. This type of data can be difficult to model since it may only be known that it occurs relative to another data value or event as opposed to having an absolute start, end, or duration time to associate with it. Associated with relative temporal relationships are those relationships in which one event preceeds, and in turn, triggers another event. Data which describes tasks used in scheduling, where the completion of one task triggers the start of another, is an example of such data. Relative temporal data does have important applications but is not the focus of this thesis. It is mentioned here only to provide a complete picture of time/data relationships.

1.3 ORGANIZATION OF THESIS

The purpose of this thesis is to discuss the most recent research in the development of a temporal data model. The requirement for temporal support in databases has been understood since the late Seventies. Two bibliographic surveys of the research in this area , "Bibliography: Temporal Databases" (McKenzie [31]) and "Researching Concerning Time in Databases: Project Summaries" (Snodgrass [41]), covered the research on time oriented databases up 1986. This thesis concentrates on the research since that time.

The research presented focuses primarily on extensions to the relational data model to support temporal data. Two additional temporal data models also covered are based on the Entity-Relationship model and a unique temporal specific data model.

The basic concepts of temporal data are discussed in Chapter 2. An introduction to temporal data is presented independent of implementation or performance concerns. The Entity-Relationship Model and its extensions to support temporal data are introduced in this chapter as well, since it is the methodology often used to present data and relationships at a conceptual level. The temporal data model includes the database objects or data structures and the operators which manipulate these objects. Chapter 3 discusses the data structures of the temporal data models based on the data structures of the relational model and those of the temporal specific data model. Chapter 4 introduces proposed language extensions to support access to and manipulation of temporal data. The physical data model, discussed in Chapter 5, describes proposed physical implementationsof the data structures of the temporal data models.

2. BASIC CONCEPTS OF TEMPORAL DATA

2.1 INTRODUCTION

Prior to examining the temporal data model in detail, it is important to examine the relationship between time and data at a conceptual level, independent of any particular data model or DBMS. As mentioned previously, the type of time/data relationship of interest in this thesis is that of data attribute values varying over time. This chapter identifies these relationships and establishes a set of terminology that will be used in further discussions of temporal data.

Section 2.2 defines several properties of temporal data which are incorporated into the temporal data models. Also in this section, the four fundamental types of databases supporting temporal data are introduced. Section 2.3 discusses the Entity-Relationship Model and the enhancements that support temporal data. This data model, based on entities and their relationships, is often used to define the data requirements and the relationships at a conceptual level in the database design process.

2.2 PROPERTIES OF TEMPORAL DATA

This section defines the properties of time related data to be incorporated into a temporal data model. Specifically, the properties of data attribute values varying over time are defined, which includes types of time sequences, regularity, time unit, static and dynamic data, data persistency, and time versioning of data.

2.2.1 Time Sequences

The concept of attribute values varying over time has been examined in depth by Shoshani and Kawagoe [39, 40]. They have termed a sequence of attribute values over time a "time sequence" and have built their temporal data model around this concept. Four distinct types of time sequences have been defined: "discrete", "continuous", "step-wise constant" and "event".

A discrete time sequence is one in which the attribute values occur and are recorded only at specific times. Each of the values recorded are independent of the other values in the sequence. An example of a discrete time sequence is the daily attendance of an employee. The number of hours worked is recorded at the end of the day, for each employee. The number of hours worked by an employee for a particular day is not related to the number of hours worked on the previous days. The values are discrete, and new values cannot be derived or interpolated from previous values in the time sequence.

A continuous time sequence is one which has a value for every point in time. In a continuous time sequence, the value of the attribute is often changing at intervals smaller than the sampling interval and thus, the attribute value can change many times between samplings. For example, the magnetic field around an electric conductor is a continuous attribute which will always have a value. The strength of the magnetic field attribute may vary many times during a one minute interval between measurings, but only the readings at the end of one minute intervals will be saved. An "approximation or derivation function" can be used to derive the data value at a point between any two sampling points.

In a step-wise constant time sequence, the value of the attribute is also continuous, but it remains constant at its current value until the next recorded time point at which it changes to a new value. The current salary of an employee is an example of a step-wise constant attribute. This attribute will remain constant at its current value until the next salary change. No interpolation function is required with this type of continuous data.

An event is something which occurs at a particular point in time. With event data, the occurrence of the event is as important as the attributes which describe the occurrence. An event time sequence is a type of step-wise constant or discrete time sequence in which the attribute domain consists of two possible values. The two possible values indicate if the event did or did not occur. In addition to measuring the magnitic field around the electrical conductor, the fact the measuring device was also functioning correctly could also have been recorded, as a binary value. This would be an example of step-wise constant event data.

2.2.2 Regularity

The "regularity" (Shoshani and Kawagoe [40]) of data over time refers to the interval of time between the recording of time/data pairs in a sequence. A "regular" sequence is one in which this interval is constant. An "irregular" sequence is one in which this interval varies or is random. The scientific database which records the strength of a magnetic field at one minute intervals is an example of a regular sequence. A series of salary increases for an employee as a result of promotions is irregular time sequence data.

2.2.3 Time Unit

The "time unit" (Shoshani and Kawagoe [40]) is the interval length between the points in time that can potentially have data values. In the magnetic field example, the time unit is one minute between samplings.

2.2.4 Static and Dynamic Data

The property of data being static or dynamic (Shoshani and Kawagoe [40]) refers to the future update activity on the data. Static data has been fully recorded and no further data updates or additions will be made to the database. An example of such a database is one which contains scientific data which has been collected over a period of time (ie. during an experiment or particular event) that will be used for analysis. A database storing dynamic data, on the other hand, can still be added to and updated. The recording of dynamic data can be stopped at some point though, and the data collected to that point can be considered a static set.

2.2.5 Data Persistency

Several papers on temporal databases discuss the length of time historic data will exist in a database. Adiba and Quang [2] refer to the "persistency of values in the databases" in reference to the length of time and number of versions of data that are stored in a database. Theoretically, a temporal data model places no restriction on the number of occurrences of a data attribute retained over the time domain. In an implementation of a temporal data model though, it may not be practical, nor required for business purposes, to retain an infinite number of occurrences.

Segev and Shoshani [39] also refer to the "life span" of data. The life span is the "range of valid time points" for the historic data collected for an entity, defined by a start and end time. They identify three types of life spans: life spans with a fixed start and fixed end point; life spans with a fixed start point and the current time as the end point; and lifespans where there is a fixed distance between the start and end points and the end point is the current time. A life span with a fixed start and fixed end point defines a static set of data. A life span with a fixed start point and the current time as the end point is a dynamic set of data which continues to grow as data is added. In a life span where there is a fixed distance between the start and

end points and the end point is the current time, only a fixed amount of data is retained and old data is dropped off as new data is added. There is, in effect, a fixed size "moving window" over the data occurrences over time.

2.2.6 Time Dimensions of Data

There are two time "dimensions" associated with data that are of interest to a user when maintaining temporal data. These two dimensions are the "transaction time" and "valid time" (Snodgrass and Ahn [43]) associated with the data. Transaction time is defined as the time the data is recorded in the database and valid time is the time the data value becomes valid in the real world. Both may be required since the valid time will not necessarily be the same as the time the data was updated in the database. This is the case when the data is changed as a result of a retroactive update, an update of a prior version of the data, or a proactive update, an update which will be applied at some point in the future, or simply because there is a delay in entering the update of the data into the database.

A third type of time attribute, "user-defined time" has been defined by Snodgrass and Ahn. A user-defined time attribute is simply an attribute in the database which contains time data, such as a date of birth. The addition of user-defined time data is discussed in detail by Date [17,18]. Unlike transaction and valid time, which are maintained by a temporal DBMS, user -defined time is application maintained and not interpreted by the DBMS. User-defined time does not influence the development of the temporal data model and will not be discussed further in this thesis.

Four fundamental types of time-related database have been defined (Snodgrass and Ahn [43], Snodgrass [42]) based on transaction and valid time. These are "snapshot", "rollback", "historical" and "temporal" databases.

A snapshot database has no DBMS support for maintaining temporal data. This is the traditional type of database which provides a "snapshot" of the data as it exists at a point in time. A snapshot is the set of data values that exists for all attributes in the database at a point in time. A snapshot will also be referred to as a database state. As data in a snapshot database is modified, the current data is updated and the prior database state is lost. The handling of any temporal data in a snapshot database must be coded into the application program using user-defined time attributes.

A rollback database is a one where past states are retained automatically. In such a database the concept of transaction time is used to associate a database state with a time. When data is modified in a rollback database, a new version of the updated data is appended to the database and becomes the current version. The previous version is retained unchanged. Updates to a rollback database can only be applied to the most current version of the data. Previous versions may not be altered, even for error correction. The rollback database can be represented as a sequence of database states (snapshot databases) ordered by transaction time. With this type of database, queries can be made against previous states of the database by "rolling back" the data to a prior snapshot and performing the query on this version of the data. Note that in the discussion of temporal databases, "rollback" refers to the operation of temporarily making a prior version of the data the current version. The major drawback of this database type is that it records the information by the time the data was modified in the database (transaction time) and not the time the event occurred in the real world. In situations where retroactive and proactive updates are being made, though, associating the data with the transaction time is not valid. To handle these situations, the historical database was developed.

The historical database is similar to a rollback database except that prior states are recorded by valid time rather than transaction time. The "valid time" is the time the data became valid in the real world. This type of database is a collection of "historical states" ordered by valid time. A historical state can be defined as a snapshot associated with a valid time, as opposed to a database state which is a snapshot associated with a transaction time. Unlike rollback databases, in which only the most current version of the data can be modified, the historical database allows updates to the current or previous data to accomodate retroactive and

proactive updates. Thus, it may not be possible to "roll back", as in a rollback database, to a version of the data which existed at a given point in time since prior versions may be changed, and the historical state which existed at that time may have since been updated with more current information. The historical database can always store the most current information about past states because retroactive updates are supported. Similar to the rollback database, a historical database allows queries to be run against past historical states by rolling back the data to this state and performing the query on the resulting data.

A temporal database combines the features of the rollback and historical databases by associating both transaction and valid time with data. This type of database is essentially a sequence of historical databases ordered by transaction time. With a temporal database, queries of the form "what was the value of attribute A at time t_0 if this query was executed at time t_1, assuming it is now time t_3" are supported. In this form of query, a roll back to the historical database at time t_1 is performed and then the historical state at time t_0 within this database is used to determine the result. A temporal database allows the user to query a temporal occurrence of the data which existed at some point in the past. In this thesis, a temporal database will be used to refer to a database which associates both transaction time and valid time with data. The term temporal data model will be used in reference to any data model which supports temporal data.

2.3 TEMPORAL DATA IN THE ENTITY-RELATIONSHIP MODEL

A conceptual representation of data and data relationships describing an enterprise is usually done with a "high-level sematic model" (Winsberg [48]). The model often used is the Entity-Relationship Model, or some variation of it. Temporal data support has been incorporated into the conceptual representation of data as enhancements to the Entity-Relationship Model. Prior to a discussion of temporal data in the Entity-Relationship Model, a brief overview of the Entity-Relationship Model itself will be presented.

2.3.1 The Entity-Relationship Model

A common approach to representing a conceptual data model is to use the Entity-Relationship (E-R) Model. As the name implies, this method identifies the entities within an enterprise and the relationships between them, independent of the processes involving these objects. An entity is any identifiable thing for which data must be maintained to support the activities of an organization. The entity may represent an object that tangibly exists, such as a person, place or thing, or a concept, such a project or event. Entities are typically represented pictorially in E-R diagrams as rectangles. A relationship defines a connection between entities representing interactions and associations between the objects. Relationships are represented in E-R diagrams as diamonds joined to the related entities with solid lines. The properties of entities and relationships are called attributes. Each entity has a primary key attribute or attributes associated with it which uniquely identifies the entity. For example, the primary key of an employee entity could be an employee number.

An example of an E-R model for part of a personnel database representing employees and their departments, is shown in Figure 2.1. The attributes of the entities are also listed with primary key fields underlined.

FIGURE 2.1 EMPLOYEE-DEPARTMENT ENTITY RELATIONSHIP MODEL

ENTITY	ATTRIBUTES
EMPLOYEE	employee number, name, position, salary, birthdate
DEPARTMENT	department number, department name, budget

RELATIONSHIP	ATTRIBUTES
EMPLOYEE/DEPARTMENT	(none)

FIGURE 2.1 EMPLOYEE-DEPARTMENT ENTITY RELATIONSHIP MODEL

2.3.2 Temporal Enhancements To The E-R Model

Enhancements to the E-R Model to support temporal data have been proposed by Klopprogge and Lockemann[26] and Adiba and Quang[2]. Though the diagramming technique associated with the E-R Model is useful component of this model, neither Klopprogge and Lockemann nor Adiba and Quang have made enhancements in this area. Instead, both have enhanced a high level definition language which has been used to represent the definition of entities and relationships. The E-R Model has been used to show how temporal data can be incorporated into a conceptual representation of data.

(A) Klopprogge and Lockemann

Klopprogge and Lockemann [26] have extended the E-R Model to develop the temporal data model TERM (Time Extended E-R Model). In their discussion of the Entity-Relationship model and the TERM data model, Klopprogge and Lockemann have referred to both entities and relationships as "objects", and that convention will be retained here.

In the TERM data model, Klopprogge and Lockemann maintain temporal data at the individual attribute level rather than at the object level. The reason for this is that some attributes within an object are "temporally constant" and do not change over time, and others are "temporally variable" and change over time. Maintaining history at the object level would mean maintaining history of constant attributes, resulting in unnecessary redundancy. The temporally varying attributes are maintained as a series of time/value pairs. The definition for the EMPLOYEE entity in the TERM data model is shown in Figure 2.2.

```
ENTITY TYPE
    employee;
  EXISTENCE
      VARIABLE;
  ATTRIBUTES
      employee number   CONSTANT  integer;
      name              VARIABLE  char(20);
      position          VARIABLE  char(10);
      salary            VARIABLE  decimal(7,2);
      birthdate         CONSTANT  date;
```

FIGURE 2.2 ENTITY DEFINITION (Klopprogge and Lockemann)

A special "existence" attribute, which is temporally variable, is also associated with each object. In this model, object history is maintained over the entire lifespan of the database. The boolean "existence" attribute is used to indicate the periods of time that the object exists in the real world. This attribute has the value of true if at least one of the attributes of the object has a defined value.

Klopprogge and Lockemann also discuss the uncertainty which may exist when recording data values over time. When an attribute is undefined at a point in time, its value is null. An example would be the social security number of a child which has not yet been assigned a number. The value of an attribute can also be "uncertain" or "unknown". An attribute can have an "uncertain" value if its value is not known at the current time, but it is known that it may have a null or non-null value, such as for the social security number of any person. An "unknown" value is assigned to those attributes whose value is not known, but it is known that the value is not null. The existence attribute may also take on an "unknown" value if it is initially not known. Procedures and logic are necessary to determine object existence and attribute values when uncertainty exists and "approximation functions" are required to handle uncertainty when continuous data is introduced.

Update semantics are also discussed by Klopprogge and Lockemann and the notion of updating versus correcting is addressed. In the TERM data model, two different types of users are defined to handle these situations. A "recorder" is a user whose updates append new temporal occurrences of data to the database while a "referee" is a user who can update existing data in the database to correct previous errors without adding a new state to the database.

The TERM data model represents a rollback database since only a single dimension of time, the transaction time, is associated with the data. A TERM prototype has been implemented by Klopprogge and Lockemann as an extension to PASCAL with an interface to a network DBMS.

(B) Adiba and Quang

Adiba and Quang [2] have also based their temporal data model, the TIGRE data model, on the Entity-Relationship Model, in an attempt to incorporate time into a generalized DBMS. In their approach, as in that of Klopprogge and Lockemann, temporal data is maintained for only one time dimension. Adiba and Quang have assumed that the difference in the time the event took place and the recording time in the database is irrelevant. Thus, their temporal data model is a historical database where temporal data is associated with a valid time.

The TIGRE data model has been developed to model complex objects which contain text, image and voice data. Unlike the TERM data model, the TIGRE data model does not maintain temporal data at the attribute level for all attribute types occurring within an entity. The basic attribute types (integer, character, etc.) cannot be selectively versioned, while the complex types (such as record and defined types) can be. In general, temporal data is maintained at object level (entity or relationship), which in turn propagates to all of the attributes of that object. If the object is a relationship and defined to maintain history, the time dimension propogates to the related objects, which must be defined as being non-temporal. The entities and relationships outlined in Figure 2.1 would be defined in the TIGRE data model as shown in Figure 2.3. In this example, the last five relationships between employees and departments will be maintained. This property will automatically propogate to the employee and department entities, which have not been defined to maintain history.

Temporal data in the TIGRE data model is based on the concepts of "periodicity", "modification" and "persistency". The notion of periodicity, in the TIGRE data model, is the length of time between values in a sequence of value/time pairs. For example, the salary values for an employee may be recorded each month for the past year. This does not imply, though, that the salary is updated each month to generate each temporal occurrence. Periodicity is the term used by Adiba and Quang for the regularity property of temporal data defined previously.

```
type EMPLOYEE : entity;
      employee number : integer;
      name  : string(20);
      position  : string(10);
      salary  : decimal(7,2);
      birthdate  : date;

type DEPARTMENT : entity;
      department number  : integer;
      department name  : string(20);
      budget  : decimal(7,2);

type EMP/DEPT : dynamic relationship last 5
                  between EMPLOYEE and DEPARTMENT;
```

FIGURE 2.3 ENTITY RELATIONSHIP DEFINITION (Adiba and Quang)

In the TIGRE data model, updates to an object or an attribute may or may not generate a new temporal occurrence of that object. That is, modifications to an object or attribute may update only the current version and the updated version is used to generate a new temporal occurrence only at the end of a defined time period. Several changes can be made before a new temporal occurrence is added. An object in the TIGRE data model can be defined as having a "manual history", "periodical history" or "successive history". In objects which have a manual history, the user must determine when a new temporal occurrence of the object is to be created. With periodical history, the length of the period between the generation of temporal occurrences is defined to the DBMS and at the end of the defined period, a week for example, the current version is automatically used as the new temporal occurrence. During the period (ie. the week), all updates will have modified the current version. An object may also be defined to have successive history in which each modification automatically adds a new temporal occurrence of the object. It is up to the database administrator to determine when an update should create a new temporal occurrence. The property of controlling the generation of temporal occurrences is called the modification property.

The persistency concept refers to the length of time historic versions are retained in the database. In theory, historic data may be retained indefinitely, but this may not be practical or required. Therefore, Adiba and Quang have placed the definition of the length of time and the number of occurrences historic data to be retained with the definiton of the object.

2.3.3 Comparison of E-R Model Enhancements

The enhancements to the E-R Model by Klopprogge and Lockemann and Adiba and Quang, are proposals to add temporal data support to a generalized data model, independent of any particular DBMS. Both approaches are similar only in that the data is augmented with a single time dimension. Klopprogge and Lockemann have developed a temporal data model based on transaction time, while Adiba and Quang have used valid time.

The rollback database model defined by Klopprogge and Lockemann supports temporal data at the attribute level. Adiba and Quang's historic database model supports temporal data at the object and complex attribute (ie. record and defined types) level, and the time dimension propogates to object attributes. The time dimension will also propogate across a relationship in Adiba and Quang's temporal data model, if the relationship is defined to maintain temporal data and the entities are defined to be non-temporal. The modification property, in which generation of temporal occurrences is controlled, is a property of temporal data unique to Adiba and Quang's model. This property is useful in that it allows the data base administrator to determine how frequently a temporal occurrence of an object should be generated, depending on the nature of the data. For example, an entity defining an entire document may be defined to have a manual history, so that only the versions of the document chosen by the user are migrated to history, rather than each version created. An employee entity though, may be defined to have a successive history so that every version of the employee information is retained.

Klopprogge and Lockemann make a useful distinction between users who do updates and those who do corrections. A rollback database does not support retroactive updates, but in their data model, a special type of user has been defined which may correct previous temporal occurrences of data. A second important contribution of Klopprogge and Lockemann is their discussion of object and existence and uncertainty with temporal data and the requirement for procedures to handle this.

2.4 SUMMARY

This chapter has defined the basic concepts and properties of temporal data required for the development of a temporal data model. These properties include the types of time sequences (discrete, continuous, step-wise continuous and event), the regularity of temporal data, static and dynamic data, and data persistency. The two time dimensions of temporal data, transaction time and valid time, have been introduced and four database types based on these dimensions have been defined. These database types include the snapshot database (no temporal data support), the rollback database (temporal data support using transaction time), the historic database (temporal data support using valid time) and the temporal database (temporal data support using transaction and valid time).

The E-R Model and enhancements to it to support temporal data were also discussed in this chapter. Though initially intended as a data model, the E-R Model and its diagramming technique are primarily used in the database design process to represent the logical structure of a particular database at a conceptual level, rather than as a data model. Enhancements to the conceptual representation level to support temporal data are significant in that they provide the ability to incorporate temporal concepts at the initial data modelling stage in a structured manner. Adding temporal data support at the entity and relationship (Adiba and Quang)

or attribute (Klopprogge and Lockemann) level permits the conceptual model to represent the temporal requirements. This simplifies the database design and implementation since the entities and attributes added strictly to support the temporal requirements of the data have been eliminated.

Having defined the basic concepts of temporal data and examined the temporal enhancements at the conceptual representation level, the next step is to proceed to the development of the temporal data model to support these requirements.

3. TEMPORAL DATA STRUCTURES

3.1 INTRODUCTION

A data model provides a definition for the structure of objects to represent data and data relationships, and a set of operations to manipulate these data structures. The relational data model, for example, uses the relation as the basic data structure and the operations of SELECT, PROJECT and JOIN to manipulate the relations. A temporal data model is one which provides support for the properties of temporal data defined in the previous chapter. This chapter defines and compares proposals for the data structure component of the temporal data model. The operations defined on these temporal data structures are discussed in the following chapter.

Most of the research on the temporal data model has taken the approach of extending the relational data model to support temporal data. Thus, the majority of this chapter will focus on enhancements to the relational data model. A data model based specifically on temporal data and time sequences has also been developed, and the data structures of this model will also be discussed in this chapter.

The organization of this chapter is as follows. Since the temporal data model is most often defined as an extension to the relational data model, a brief overview of the relational data model is presented, and the four database types supporting temporal data (snapshot, rollback, historic, and temporal) are presented in terms of relations. This is followed by a discussion of some issues to be addressed in enhancing the relational data model, including maintaining temporal data at the tuple or attribute level, using time instant or time interval timestamps, and handling schema modifications. A comparison of proposed temporal data models is then presented. The approaches are covered in four sections: temporal data models based on application maintained data; temporal data model based on the relational model with tuple versioning; temporal data

model based on the relational model with attribute versioning; and the temporal data specific data model.

3.2 THE RELATIONAL DATA MODEL

The relational data model, initially proposed by E. F. Codd [13], is based on the mathematical theory of sets. In this model, a relation is a collection of tuples, where each tuple element, or attribute, *s,* is from some domain *S.* The ordering of the tuples within a relation is not defined but the ordering of the attributes is fixed. The primary key of a relation is an attribute or attributes which can be used to uniquely identity a tuple in a relation. Relations can be represented as two dimensional tables as in Figure 3.1. This is not to imply though, that they are physically stored in this manner. In addition, the data is manipulated in terms of relations and tuples, removing the user from the physical organization and implementation of the data.

EMPNO	EMPNAME	SKILLS
0100	AAAAA	1
0100	AAAAA	4
0200	BBBBBB	3
0305	CCCCCC	1
0305	CCCCCC	5

SKILL	DESCRIPTION
1	COBOL
2	PL/1
3	C
4	JCL
5	DB2

FIGURE 3.1 FIRST NORMAL FORM RELATIONS

EMPNO	EMPNAME	SKILLS
0100	AAAAA	[1,4]
0200	BBBBBB	[3]
0305	CCCCCC	[1,5]

SKILL	DESCRIPTION
1	COBOL
2	PL/1
3	C
4	JCL
5	DB2

FIGURE 3.2 NON-FIRST NORMAL FORM RELATION

In Codd's relational data model, all relations must be in at least "first normal form". A relation is said to be in first normal form if all the attributes in the relation are "atomic". Atomic attributes are those which can not be further subdivided, and are assigned a single value at any point in time. This is as opposed to a "set-valued" attribute (Abbod, Brown and Noble [1]), in which a set of data values is assigned to an attribute. An example of a set-valued attribute is an attribute storing the skills of an employee. As new skills are acquired by the employee, the existing skills are not replaced, but instead are grouped with the new data to form a set of skills for the employee (Figure 3.2). Set-valued attributes are not allowed in first normal form relations, but there are temporal data models based on extensions to the relation model which do. Higher levels of normal forms, third normal form and higher, have been developed which further remove data redundancies from first normal form relations in order to provide equivalent (ie. no loss of information) and more desirable relations for implementation.

Another type of non-first normal form relation which has been proposed, and which a temporal data model has been based on, is a nested relation. A nested relation is one where, in addition to atomic and set-valued attributes, relations are permitted as attributes. (Figure 3.3) Nesting relations in this manner introduces a hierarchical relationship between the data in the relations. It has been suggested [19,32] that "flat relations" (first normal form) are not ideal for modelling complex objects and relationships. For example, in Figure 3.3, the entire skills relation is incorporated into the employee relation to maintain the set of skills for an employee. One reason suggested for this is that in the normalization process, the data is unneccesarily

fragmented among several relations, both physically and logically, resulting in queries that must access several tables in order to retrieve the required data.

EMPNO	EMPNAME	SKILLS	
		SKILL	DESCRIPTION
0100	AAAAA	1	COBOL
		4	JCL
0200	BBBBBB	3	C
0305	CCCCCC	1	COBOL
		5	DB2

SKILL	DESCRIPTION
1	COBOL
2	PL/1
3	C
4	JCL
5	DB2

FIGURE 3.3 NESTED RELATION

3.3 TIME ORIENTED RELATIONS

In the previous chapter, the four types of databases supporting temporal data, based on transaction and valid time, were introduced. These were the snapshot, rollback, historical and temporal databases. In this section, these database types will be presented in the context of relations, the basic data structure of the relational data model.

A snapshot relation is the classical relation comprising a collection of tuples consisting of atomic attribute values. This type of relation provides a "snapshot" of the data as it exists at a point in time. A snapshot relation is typically represented as shown in Figure 3.4. This type of relation can be referred to as "two dimensional", where the two dimensions are the rows and the columns.

EMPNO	EMPNAME	POSITION	SALARY	BIRTHDATE	DEPTNO
0100	Adams	Manager	50,000.00	1933-05-01	A00
0605	Thompson	Clerk	21,000.00	1969-08-04	A00
0200	Spencer	DBA	30,000.00	1963-09-09	A00
0300	Jones	Analyst	29,000.00	1962-12-07	A00
0305	Brown	Programmer	26,000.00	1866-08-08	A00

FIGURE 3.4 SNAPSHOT RELATION

A rollback relation is one in which past states of a relation are retained and associated with a transaction time. When an update is made to a rollback relation, a new snapshot of the relation is appended to the rollback relation along with the transaction time. This newest snapshot becomes the current version, and the previous snapshots are retained unchanged. A rollback relation can be represented as a sequence of snapshot relations ordered by transaction time (Figure 3.5). The transaction time becomes a "third dimension" in the representation of this relation.

EMPNO	SALARY
0100	50,000.00
0605	21,000.00
0200	30,000.00
0300	29,000.00
0305	26,000.00

EMPNO	SALARY
0100	50,000.00
0605	21,000.00
0200	30,000.00
0300	29,000.00
0305	26,000.00
0306	25,000.00

EMPNO	SALARY
0100	50,000.00
0605	21,000.00
0200	30,000.00
0300	29,000.00
0305	33,000.00
0306	25,000.00

89/12/31 90/01/01 90/02/01

Transaction Time

FIGURE 3.5 ROLLBACK RELATION

The historical relation is similar to the rollback relation except that prior states are recorded by valid time rather than transaction time. This type of relation (Figure 3.6) is a collection of "historical states" in which valid time is the third dimension of the relation. A historical state is a snapshot of the data associated with a valid time. A historical state differs from a snapshot in a rollback relation in that a historical state can change, where a snapshot cannot. As mentioned in the previous chapter, historical relations allow the user to modify data at any time in the relation.

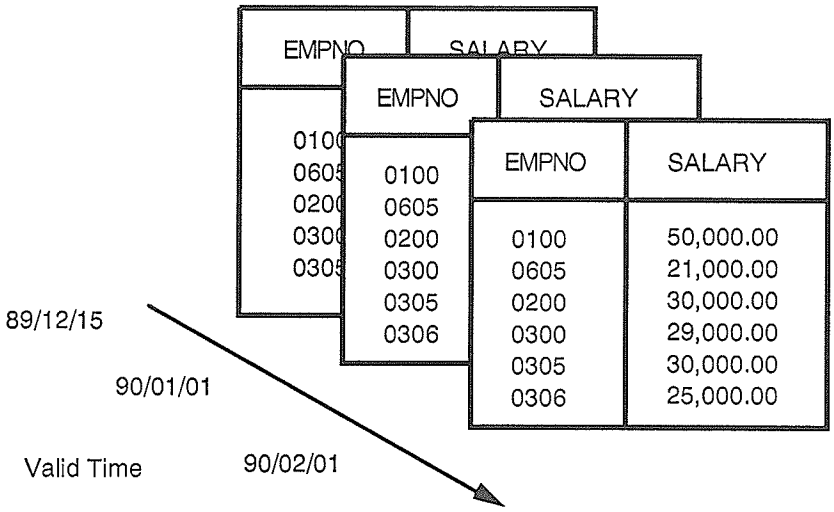


FIGURE 3.6 HISTORICAL RELATION

A temporal relation combines the features of the rollback and historical relations by incorporating both transaction and valid time. This "four dimensional" relation is essentially a sequence of historical relations indexed by transaction time (Figure 3.7). In a temporal query, it is possible to roll back to a prior historical relation and then specify a particular historical state within this relation. A query against a temporal relation allows the user to retrieve the data which is current as of some point in the past. When an update is performed, a new historical relation is created and appended to the existing temporal relation.

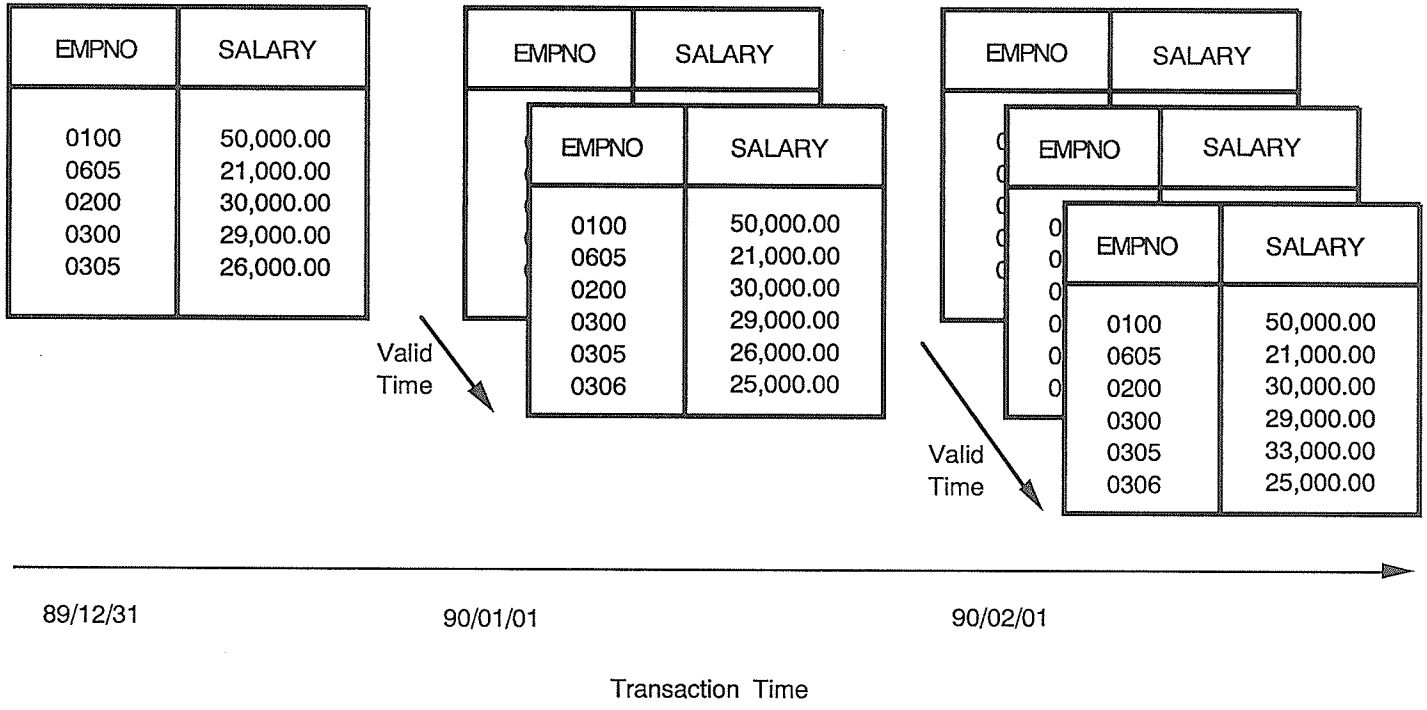


FIGURE 3.7 TEMPORAL RELATION

3.4 ISSUES IN ENHANCING THE RELATIONAL DATA MODEL

There are several ways in which the relational data model can be enhanced to support temporal data. The key issues in determining how the relational model should be enhanced are introduced here to lay the groundwork for the temporal data models discussed in the next sections. The key issues include the level at which temporal data should be maintained, the type of timestamp that should be used to represent transaction and valid times, and how schema modifications should be handled.

3.4.1 Tuple and Attribute Versioning

As described above, time-oriented relations maintain previous states or versions of data, indexed by valid and/or transaction times. Though the preceeding discussion of temporal relations and their associated pictorial representation presented versioning at the relation level for ease of understanding, the versioning is more appropriate at the tuple or attribute level since this is the component of the relation manipulated by the update and query operations. It has been suggested by Gadia[23,24] and Sarda[35] that since the tuple is the basic unit of manipulation of query languages, the versioning should not occur at a level higher than the tuple level.

In tuple versioning, when the value of an attribute changes, a new version of the entire tuple is created. In attribute versioning, when the value of an attribute changes, a new version of only that attribute is created. Ahn[3] states that relations which use attribute and tuple versioning contain the same information and operations can be defined to convert from one type to the other. Ahn has defined the UNNEST and SYNCH functions to perform these operations. The UNNEST function converts a relation using attribute versioning to one which uses tuple versioning and SYNCH transforms a tuple versioning relation to one which uses attribute versioning.

All temporal data models reviewed use either attribute or tuple versioning; each has advantages and disadvantages. The benefit of attribute versioning is derived from the difference in the rate of update of the attributes in a tuple. With attribute versioning, when an attribute in a tuple is updated, it is necessary only to create a new version for the changed attribute. Therefore, it is possible for only selected attributes within the tuple to be temporal. The versioning independence between attributes is not only beneficial for temporal attributes which vary at different rates, but also for those attributes in a tuple which remain constant and will never vary over time. Attribute versioning will eliminate the duplication of constant information which occurs when tuple versioning is used. Dadam, Lum, and Werner [14] have termed this "selective versioning".

The main factor favoring tuple versioning is that it results in normalized relations. Relations having versions of attributes within a tuple are not in first normal form since the temporal attributes are set valued and are assigned a series of time/value pairs. The fact that the relations are not normalized is significant since the properties of the relational data model and relational operations are based on first normal form relations. The unfamiliarity and lack of formalization of non-first normal form relations could add complexity to building a temporal data model based on this model. This may be the more desirable form though, since temporal data is very naturally set oriented over attributes. In fact, Gadia [23] states that the direction for temporal databases is toward non-first normal form relations.

In order to eliminate non-first normal form relations and still retain the capability to selectively version attributes, the temporal attributes could be placed into separate relations each containing the snapshot primary key and a single temporal attribute. Each of the resulting relations can then use tuple versioning and still remain normalized. Note that, when discussing relations in a temporal data model, the snapshot primary key refers to an attribute, or attributes, which uniquely identify a tuple in a snapshot from a relation supporting temporal data. In the rollback, historical and temporal relation presented in Figures 3.5, 3.6 and 3.7, the snapshot primary key is the employee number, EMPNO.

Gadia [23,24] has defined two "anomalies" which occur as a result of using this approach. These have been termed the "vertical and horizontal temporal anomalies".

Consider as an example a relation with two temporal attributes, POSITION and SALARY. The "vertical temporal anomaly" occurs when a tuple with a given snapshot primary key is split up into multiple tuples with that key when a time varying attribute is updated. (Figure 3.8) In this example, when an employee's POSITION or SALARY changes, a new tuple for that employee is created. Thus, the number of tuples in the temporal relation for a snapshot primary key in the relation is increased; the relation size is increased "vertically". In order to retrieve all the information for an employee, multiple tuples must be accessed.

EMPNO	POSITION	SALARY	TIMESTAMP
0100	Manager	50,000.00	1990-01-01: NOW
0605	Clerk	21,000.00	1990-01-01: NOW
0200	DBA	30,000.00	1990-01-01: NOW
0300	Analyst	29,000.00	1990-01-01: NOW
0305	Programmer	26,000.00	1990-01-01: 1990-02-01
0305	Analyst	33,000.00	1990-02-01: 1990-03-01
0305	Analyst	30,000.00	1990-03-01: NOW
0306	Programmer	25,000.00	1990-01-01: 1990-05-01

FIGURE 3.8 VERTICAL TEMPORAL ANOMALY

To minimize the number of tuples for a snapshot primary key and still maintain first normal form relations, the relation could be split into two relations, one containing the temporal position attribute and one containing the temporal salary attribute (Figure 3.9). This results in what Gadia calls the "horizontal temporal anomaly". The "horizontal temporal anomaly" occurs when the attributes of a tuple are split up into multiple relations to support multiple temporal attributes that change at different rates. In order to access all data for an employee, multiple relations have to be accessed. In this case, the data has been spread out "horizontally" over multiple relations.

EMPNO	POSITION	TIMESTAMP
0100	Manager	1990-01-01: NOW
0605	Clerk	1990-01-01: NOW
0200	DBA	1990-01-01: NOW
0300	Analyst	1990-01-01: NOW
0305	Programmer	1990-01-01: 1990-02-01
0305	Analyst	1990-02-01: NOW
0306	Programmer	1990-01-01: 1990-05-01

EMPNO	SALARY	TIMESTAMP
0100	50,000.00	1990-01-01: NOW
0605	21,000.00	1990-01-01: NOW
0200	30,000.00	1990-01-01: NOW
0300	29,000.00	1990-01-01: NOW
0305	26,000.00	1990-01-01: 1990-02-01
0305	33,000.00	1990-02-01: 1990-03-01
0305	30,000.00	1990-03-01: NOW
0306	25,000.00	1990-01-01: 1990-05-01

FIGURE 3.9 HORIZONTAL TEMPORAL ANOMALY

Thus, when attempting to preserve first normal form relations and make selective attributes temporal, there is a trade-off between the "horizontal anomaly" and "vertical anomaly". Reducing one will increase the other. In order to eliminate these anomalies, Gadia uses non-first normal form relations for his temporal data model rather than fitting the temporal data model into the normalized relations of the relational data model. In the development of the various temporal data models, approaches based on attribute and tuple versioning have been proposed.

3.4.2 Timestamps

Timestamps are required in a temporal data model to distinguish between temporal versions of data. As discussed previously, at most two types of timestamps are required, independent of whether tuple or attribute timestamping is used. These two timestamps represent the "transaction time" and the "valid time". They are also referred to as "physical" and "logical" timestamps, respectively. The transaction time is the time the data was physically added to the database. Valid time is the time the data logically became valid in the "real world". When managing "temporal data", as defined by Snodgrass [43], both timestamps are required. For "historical databases" only valid time is required and "rollback databases" record only transaction time.

An important aspect of timestamps is whether they should represent a time instant or a time interval. If the timestamp represents a time instant, then it indicates the time at which the value was acquired. If the timestamp represents a time interval, then it indicates the period of time that the value applies. The time interval for an attribute's value is important for temporal queries since the result set depends on when a value was acquired and when it was replaced by another. If time instant timestamps are used, two successive versions of tuples or attributes have to be examined to determine the period of validity, adding additional complexity to the implementation of the query language. One disadvantage of using interval timestamps though, is that the second time value (end time) is not usually known when the timestamp is created, and thus it is necessary to update this value at some later point in time. This requirement for update would eliminate the possibility of using a write-once storage device, such as optical disks (Abbod, Brown and Noble [1]), which can store large volumes of data at a lower cost per byte than magnetic storage. In most of the temporal data models, interval timestamps have been used and the write-once update problem has been eliminated by using write-once storage devices only for data in which both timestamp values are known. The underlying assumption of the above discussion is that an attribute is atomic and the period of validity of successive values of will not overlap.

3.4.3 Schema Modifications

Another issue to consider when maintaining temporal data is the handling of modifications to the database schema. Modifications to the database structure should still allow access to historic data described by the old schema, which is still historically valid.

As data evolves over time, there are several possible modifications that could be made to a database and should be supported. These include the addition of new attributes, the change of a non-time varying attribute to a time-varying one and vice versa, and the addition of new relations. The addition of new relations can cause problems if referential integrity is enforced by the DBMS and the new relation is related to existing relations. This problem can be further complicated if there is a time dimension associated with the relations or the referential integrity rules.

Much less has been written on the evolution of database schemas than on temporal data, but from the current literature, two approaches have been proposed. The first appoach, and more intuitive one, is to treat the database catalog itself as a temporal database (Dadum, Lum and Werner [14], Adiba and Quang [2], Clifford and Croker [11]). The database catalog is a collection of system relations used by the DBMS to manage its objects, including the relations. Included in the catalog is the schema for the relations. When the schema for a relation is modified, a new version of the schema is added to the catalog. When a query is made against the data, in addition to checking to see if the tuple existed at the time in question, the lifespan of the attribute is checked in the catalog to determine if it existed at the time required. In this approach, deleted attributes are never physically removed from a relation, only from the schema definition in the catalog. New tuples are assigned null values for the deleted attributes, and inserted into the same physical relation.

A second approach has been proposed by Martin, Navathe and Ahmed [30]. In this approach, when a relation is modified, a new relation with the new schema is created but none of the data from the previous

version of the relation is copied over. In order to access the data, a query language has been developed which transparently translates temporal queries against the current relation into queries against the current and previous relations and combines the resulting tuples with the union operation to provide the solution for the original query.

3.5 COMPARISON OF TEMPORAL DATA MODELS

Four approaches to developing a temporal data model have emerged in the current research. The approaches have been grouped in this section by the modifications that have been made to the relational data model rather than by the type of temporal data that is supported. The four approaches are:

(1) the relational data model with time attributes added for application program use
(2) the relational data model with time attributes added for DBMS use
(3) the relational data model with non-first normal form relations
(4) a new data model mapped into the relational data model

In this section, the research based on these four approaches will be discussed and compared. Each section will introduce the proposed approach and then compare the specific models which use this method. In order to compare the temporal data models, the following example relations, based on the Employee-Department Entity Relationship model introduced in Chapter 2 (Figure 2.1), will be presented in each of the data models.

The EMP and DEPT relations store the data for the EMPLOYEE and DEPARTMENT entities. No relation is used to represent the EMPLOYEE/DEPARTMENT relationship since it is assumed that an employee is assigned to one department at a time, and therefore, the department number can be added to the EMP relation. The two relations with sample data at the initial state are shown in Figure 3.10. The example

also includes a series of update transactions to show evolution of the data over time (Figure 3.11).

EMPNO	EMPNAME	POSITION	SALARY	BIRTHDATE	DEPTNO
0100	Adams	Manager	50,000.00	1933-05-01	A00
0605	Thompson	Clerk	21,000.00	1969-08-04	A00
0200	Spencer	DBA	30,000.00	1963-09-09	A00
0300	Jones	Analyst	29,000.00	1962-12-07	A00
0305	Brown	Programmer	26,000.00	1966-08-08	A00

DEPTNO	DEPTNAME	BUDGET
A00	Information Systems	1,000,000.00
B00	Personnel	750,000.00
C00	Accounting	500,000.00

FIGURE 3.10 EXAMPLE RELATIONS

	Transaction	Transaction Time	Valid Time	
1.	Add EMPNO=0306	1990-01-01	1990-01-01	
2.	Update EMPNO=0305 to POSITION='Analyst' SALARY=33,000.00	1990-02-01	1990-02-01	
3.	Update EMPNO=0305 to SALARY=30,000.00	1990-03-01	1990-02-01	(retroactive)
4.	Add EMPNO=0307	1990-04-01	1990-09-01	(proactive)
5.	Delete EMPNO=0306	1990-05-01	1990-05-01	

FIGURE 3.11 EXAMPLE TRANSACTIONS

3.5.1 The Relational Data Model With Time Attributes for Application Program Use

The first approach to developing a temporal data model is to build it "on top" of the existing relational model. In this approach, tuple versioning is "implemented" by adding timestamp attributes to relations to represent valid and/or transaction times. In this approach, the underlying DBMS is not modfied in any way to interpret these time attributes as special attributes in order to manage the data. The manipulation of the temporal data is done by adding application program code to translate temporal queries into the standard queries on the underlying relations.

The reason for developing a data model using this approach is to provide a baseline for temporal database research. Such a model will point out the shortcomings of using a conventional DBMS to support temporal data, in terms of access methods, data structures and query processing algorithms. These models are not intended to be a final implementation of a temporal data model, but an intermediate step in the development of a fully integrated temporal DBMS.

The choice to build "on-top" of an existing relational DBMS, forces the use of tuple versioning. Recall that attribute versioning requires the user of set-valued attributes, which are not permitted in first normal form relations.

One advantage of the "on-top" approach, and the initial objective of Abbod, Brown and Noble [1], was that it may be possible to develop a temporal database front-end which can be easily ported to several DBMSs. Another advantage of the "on-top" approach is that a temporal DBMS can be implemented quickly since it will require no changes to the DBMS itself. In effect, an additional layer of application program code, an interface program, is placed between the user application program and the DBMS.

The major disadvantage of the "on-top" approach is performance and consistency. Since the DBMS is not aware of which tuples are current and which are historic, all tuples will be treated the same. Thus, as the

number of tuples increase, the performance will degrade, simply because the size of the relation and indexes are growing. Assuming the timestamp is not part of the primary key, there will be multiple versions of a tuple for a snapshot primary key, and conventional access methods and storage structures will quickly result in inadequate performance. In addition, all program access to the data must manipulate the temporal data consistently.

This approach to developing a temporal data model by adding time attributes to relations which are not interpreted by the DBMS has been taken by (a) Snodgrass and Ahn, and (b) Abbod, Brown and Noble.

(A) Snodgrass and Ahn

Snodgrass [42] and Snodgrass and Ahn [43] have taken the approach of adding timestamps to a tuple in a relation to essentially "embed" a temporal relation in a snapshot relation. Snodgrass [42] has proposed several methods of embedding the temporal relation in the snapshot relation. The method chosen was to append two attributes to each tuple, one for valid time and one for transaction time (Figure 3.12).

EMPNO	POSITION	SALARY	VALID TIME	TRANSACTION TIME
0100	Manager	50,000.00	90-01-01: NOW	90-01-01: NOW
0300	Analyst	29,000.00	90-01-01: NOW	90-01-01: NOW
0305	Programmer	26,000.00	90-01-01: 90-02-01	90-01-01: NOW
0305	Analyst	33,000.00	90-02-01: NOW	90-02-01: 90-03-01
0305	Analyst	30,000.00	90-02-01: NOW	90-03-01: NOW
0306	Programmer	25,000.00	90-01-01: 90-05-01	90-01-01: 90-05-01
0307	Programmer	25,000.00	90-09-01: NOW	90-04-01: NOW

FIGURE 3.12 TEMPORAL RELATION (Snodgrass and Ahn)

In this model, a distinction has been made between "event relations" and "interval relations". An "event

relation" contains data which describes discrete or event time sequences generated as a result of the occurrences of instantaneous events, such as the number of automobiles sold per day by type of automobile. In event relations, the timestamp attributes contain a single time value representing the time of the occurrence of the event. "Interval relations", on the other hand, contain data from a continuous or step-wise constant time sequence, such as the employee data shown in Figure 3.12. In interval relations, each time attribute contains two time values, a start and stop time, to represent an interval.

Thus, Snodgrass has proposed a temporal data model representing a temporal database, supporting both the valid and transaction time dimensions.

(B) Abbod, Brown and Noble

Abbod, Brown and Noble [1] have also proposed a temporal data model built on top of an existing relational model.

In their data model, each relation is augmented with a logical timestamp, a physical timestamp, a version number, a correction number and explanation attributes to handle temporal data. (Figure 3.13).

EMPNO	POSITION	SALARY	LOGICAL TIME	PHYSICAL TIME	VERS NUM	CORR NUM	EXPLAN
0100	Manager	50,000.00	90-01-01	90-01-01	0	0	
0300	Analyst	29,000.00	90-01-01	90-01-01	0	0	
0305	Programmer	26,000.00	90-01-01	90-01-01	0	0	
0305	Analyst	33,000.00	90-02-01	90-02-01	1	0	2,3
0305	Analyst	30,000.00	90-02-01	90-03-01	1	1	3
0306	Programmer	25,000.00	90-01-01	90-01-01	0	0	
0306	Programmer	25,000.00	90-05-01	90-05-01	1	0	
0307	Programmer	25,000.00	90-09-01	90-04-01	0	0	

FIGURE 3.13 SIS-BASE RELATION (Abbod, Brown and Noble)

The logical and physical time attributes store the valid and transaction times respectively. Unlike Snodgrass' model, which may store time intervals in the timestamp attributes, the timestamps used here are single time values indicating when the data in the tuple became valid, for both the logical and physical timestamps. An end time has not been included, since an update to the tuple would be required after it has initially been written to the database. This makes it impossible to use a write-once storage device to store all current and historic data.

The version number is an integer value used to indicate the position of the tuple in the logical timestamp (ie. historical) sequence of tuples. A "version update" then, is the addition of a new version to the "end" of the database. Using the version numbers, a sequence of tuples can easily be traversed in historical sequence without having to compare the logical timestamps stored in each tuple.

The correction number is used to indicate a sequence of corrections to a tuple as a result of correction updates. When a correction update is made, a correction to an existing tuple is made by inserting a new tuple with the corrected data and a logical timestamp the same as the tuple being corrected.

The explanation field indicates which attribute has been changed. An indicator of which attribute has changed as a result of a version update is required in order to handle the propagation of correction updates to subsequent versions. When a correction update modifies an attribute value, the new value must be propagated to subsequent versions until the next time that attribute is changed by a version update.

Abbod, Brown and Noble have implemented a temporal DBMS, called SIS-BASE, based on a modified version of their temporal data model . The SIS-BASE system is built on INGRES and each relation is augmented with logical time, physical time, status, pointer and explanation attributes. The status field indicates if this tuple is "initial" (the first in a series of logical versions), "terminal" (the last in a series of logical versions) or "historical" (has a version preceeding and following). The pointer attribute indentifies the previous version of this tuple. This implementation does not support correction updates; each update is

treated as the addition of a new version.

The SIS-BASE DBMS based on the temporal data model developed by Abbod, Brown and Noble represents a temporal database since both the logical (valid) and physical (transaction) dimensions of time are supported.

Summary

In summary, two proposals have been made for implementing a temporal data model on top of the unmodified relational data model. Both add valid and transaction timestamp attributes to the relations and support temporal databases. Abbod, Brown and Noble [1] learned through the implementation of SIS-BASE that adding data such as a version number, correction number and explanation field to each relation simplified the manipulation of the tuples by eliminating the need to compare the logical timestamps for each tuple when searching historical versions. In addition, their use of a version number and instant timestamps allowed the use of write-once storage devices since a tuple was not modified after it was written. The primary findings of Snodgrass and Ahn's prototype implementation were in the area of the performance of data access. They found that when the standard hashing and indexed access methods of INGRES were used, performance degraded rapidly for both temporal and non-temporal queries as the number of historic versions increased. They suggest new access methods would be required to improve performance.

3.5.2 The Relational Data Model With Time Attributes for DBMS Use

The second approach to developing a temporal data model is similar to the previous approach in that tuple versioning is used and is "implemented" by adding special time attributes to each tuple to support time varying data. This maintains the relations in first normal form. The approach differs from the previous one

in that the DBMS itself is now modified to interpret and manipulate the added time attributes, and the application programs cannot directly update these timestamps. This approach is the basis of the temporal data models developed by (a) Clifford and Warren, (b) Ariav, (c) Sarda, (d) Clifford and Croker, and (e) Rowe and Stonebraker.

(A) Clifford and Warren

Clifford and Warren [12] have incorporated time into the relational data model to provide "historical database" support by timestamping tuples. In their approach, two attributes, STATE and EXISTS? are added to each tuple. (Figure 3.14) These are special attributes which are interpreted and manipulated by the DBMS, not directly by the application program.

The STATE attribute contains a timestamp indicating the time at which a particular snapshot of the relation existed. The key of a relation in this model is the STATE attribute concatenated with the original primary key of the relation. In Figure 3.14, the primary key is a concatenation of the STATE and EMPNO attributes.

The EXISTS? attribute is a boolean attribute which has been added to create a "completed relation". A "completed relation" is a collection of snapshot relations in which each snapshot contains a tuple for every primary key that existed in the entire history of the relation. The snapshot for each STATE will contain the same number of tuples. The EXISTS? attribute is used to indicate if a tuple with a given primary key logically existed at that time. If a tuple with a given primary key does not logically exist for some state, the EXISTS? attribute will be false and all other attributes in the tuple will have a null value. For example, Figure 3.14 indicates that employee 0307 did not exist in the initial state but has been added at a later time. The concept of EXISTS? has been added to allow the model to provide the response that employee 0307 did not work for the corporation at 1990-01-01 rather than there is no such employee.

STATE	EXISTS	EMPNO	POSITION	SALARY
90-01-01	1	0100	Manager	50,000.00
90-01-01	1	0300	Analyst	29,000.00
90-01-01	1	0305	Programmer	26,000.00
90-01-01	1	0306	Programmer	25,000.00
90-01-01	0	0307	-	-
90-02-01	1	0100	Manager	50,000.00
90-02-01	1	0300	Analyst	29,000.00
90-02-01	1	0305	Analyst	33,000.00
90-02-01	1	0306	Programmer	25,000.00
90-02-01	0	0307	-	-
90-03-01	1	0100	Manager	50,000.00
90-03-01	1	0300	Analyst	29,000.00
90-03-01	1	0305	Analyst	30,000.00
90-03-01	1	0306	Programmer	25,000.00
90-03-01	0	0307	-	-
90-04-01	1	0100	Manager	50,000.00
90-04-01	1	0300	Analyst	29,000.00
90-04-01	1	0305	Analyst	30,000.00
90-04-01	1	0306	Programmer	25,000.00
90-04-01	1	0307	Programmer	25,000.00
90-05-01	1	0100	Manager	50,000.00
90-05-01	1	0300	Analyst	29,000.00
90-05-01	1	0305	Analyst	30,000.00
90-05-01	0	0306	-	-
90-05-01	1	0307	Programmer	25,000.00

FIGURE 3.14 "HISTORICAL" RELATION (Clifford and Warren)

Clifford and Warren do not suggest an implementation for their model but realize that directly implementing the fully specified "completed relation" would be very redundant and impractical.

Clifford and Warren's temporal data model supports only a single time dimension with the STATE attribute. No specification is made of the type of timestamp that the STATE attribute contains, either a valid or transaction time, so it is not possible to classify these relations as either historical or rollback relations.

(B) Ariav

Earlier in this chapter, the historical relation was represented pictorially as a three dimensional cube where the rows and columns are the first two dimensions of the relation and time is the third dimension (Figure 3.6). In Ariav's [7] "Temporally Oriented Data Model" or TODM, the data cube is used as the basic construct of his temporal data model.

In the TODM, a tuple may contain one or more "time-related attributes" (TRAs), which contain time related data. For example, the BIRTHDATE in the example EMP relation is a TRA. TRAs of this type have previously been referred to as user-defined time attributes. A particular TRA can also be defined as a "time-stamp attribute" (TSA), which is to be interpreted by the DBMS either as a transaction time or valid time to support time-varying data.

Each tuple in the TODM has a special time-stamp attribute which stores the transaction time. Ariav calls this attribute the "recording time" (RT) and it is generated automatically when a tuple is stored in a relation. In Ariav's model, it is possible to have additional time-stamp attributes to record the valid time associated with the data in a tuple. Thus, Ariav's TODM represents a temporal database.

A unique aspect of Ariav's TODM is its flexibility in defining the number of timestamps required. In addition to the transaction time which is always associated with a tuple, it is possible to have additional time-stamp attributes representing additional time dimensions in a tuple. Ariav makes no restriction on the number of timestamp attributes in a tuple, but since a tuple will be generated as a result of a single event, generally there can only be a single transaction and valid time associated with it (Figure 3.15a). Ariav has proposed a situation where multiple timestamp attributes may be used. In the example, a work order

EMPNO	POSITION	SALARY	VALID TIME	RECORDING TIME
0100	Manager	50,000.00	90-01-01: NOW	90-01-01: NOW
0300	Analyst	29,000.00	90-01-01: NOW	90-01-01: NOW
0305	Programmer	26,000.00	90-01-01: 90-02-01	90-01-01: NOW
0305	Analyst	33,000.00	90-02-01: NOW	90-02-01: 90-03-01
0305	Analyst	30,000.00	90-02-01: NOW	90-03-01: NOW
0306	Programmer	25,000.00	90-01-01: 90-05-01	90-01-01: 90-05-01
0307	Programmer	25,000.00	90-09-01: NOW	90-04-01: NOW

FIGURE 3.15a TEMPORALLY ORIENTED DATA MODEL (Ariav)

representing maintenance to an airplane may have associated with it timestamps recording the time the work was performed, the time work was inspected and the time the work order was entered in the database (ie. the transaction time). In this case, the time the work was performed and the inspection times represent two valid timestamps associated with the work order.

The "data cube" in the TODM is a collection of snapshot relations ordered by a timestamp attribute. The cube in this model is conceptually arranged in such a way that new versions of tuples are placed above the preceeding version for that same primary key (Figure 3.15b). In this way, it is possible to view the recorded history of a tuple by taking a vertical slice of the cube. This implies that the order of the tuples in the relation must remain fixed so that corresponding tuples will align in the cube data structure. This is in contradiction to the definition of a relation which states that a relation is an unordered set of tuples. Ariav justifies this by saying that the order of the tuples in the relation does not matter, as long as they remain in that order over time.

The versions of tuples in a relation in this model are by default ordered by the transaction time. Ariav also allows the definition of a type of view on the data cube which presents selected tuples from the cube ordered by a timestamp attribute other than the transaction time. Ariav calls this type of view a "temporal

interpretation with respect to T", where T is a timestamp attribute. This type of view still presents the data in a data cube form. A view on the data cube allows the definition of a "sub-cube" of the original data cube, presented in an alternate time sequence.

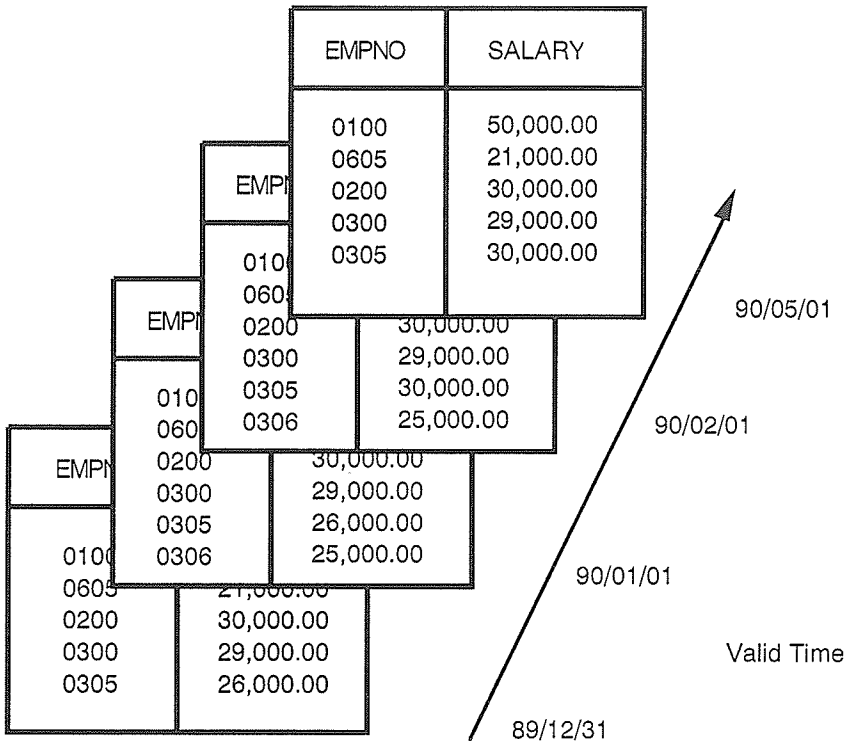


FIGURE 3.15b "DATA CUBE" BASED ON VALID TIME (Ariav)

(C) Sarda

The temporal data model developed by Sarda [35,36,37] also extends the relational data model by timestamping tuples to support historic data. Sarda states that it is important to maintain normalized relations in the temporal data model to achieve data independence and control redundancy. By timestamping tuples of normalized relations, first normal form is maintained. A second reason presented by Sarda for

timestamping at the tuple level is that conceptually, the tuple is the logical unit of manipulation by the query languages. The state of a tuple is the value of all its attributes at a point in time, therefore, timestamping should be done at the tuple level.

The timestamp used in this model is an interval timestamp representing the period of time a tuple is valid. This timestamp is intended to be a "real-world" or valid time, not the time of the update transaction (Figure 3.16). Sarda has based the decision to record only a valid timestamp on the assumption that this is the time the users are most interested in, and the difference between transaction and real world time is negligible. Since this data model is based on valid timestamps, retroactive and future updates are allowed, based on the time period specified in the UPDATE statement.

EMPNO	POSITION	SALARY	PERIOD
0100	Manager	50,000.00	90-01-01: NOW
0300	Analyst	29,000.00	90-01-01: NOW
0305	Programmer	26,000.00	90-01-01: 90-02-01
0305	Analyst	30,000.00	90-02-01: NOW
0306	Programmer	25,000.00	90-01-01: 90-05-01
0307	Programmer	25,000.00	90-09-01: NOW

FIGURE 3.16 HISTORICAL RELATION (Sarda)

Thus, Sarda's temporal data model supports only historical relations since only a single time dimension, the "real world" or valid time, is maintained.

(D) Clifford and Croker

Clifford and Croker [11] have extended the relational data model to support temporal data based on the concept of "lifespans". A "lifespan" is defined as a set of time periods, not necessarily contiguous, in

which information is stored about an entity. The concept of lifespans is considered important in a historical database since an entity may be "reincarnated" at some point. For example, a fired employee may be rehired at a later point in time.

The lifespan of a tuple with a given primary key in a relation is maintained by timestamping tuples with a timestamp attribute (Figure 3.17). Whether the timestamp attribute is a time point or time interval is not specifed. In Clifford and Croker's data model a tuple with a given primary key exists for each point in the lifespan of the tuple. In Figure 3.17, employee 0100 has a lifespan of 90-01-01 to 90-05-01, and has a tuple for each point in time in this lifespan. This is similar to Clifford and Warren's completed relation [12], but once again, this is not intended to be implemented in this manner.

EMPNO	POSITION	SALARY	TIMESTAMP
0100	Manager	50,000.00	90-01-01
0305	Programmer	26,000.00	90-01-01
0306	Programmer	25,000.00	90-01-01
0100	Manager	50,000.00	90-02-01
0305	Analyst	30,000.00	90-02-01
0306	Programmer	25,000.00	90-02-01
0100	Manager	50,000.00	90-03-01
0305	Analyst	30,000.00	90-03-01
0306	Programmer	25,000.00	90-03-01
0100	Manager	50,000.00	90-04-01
0305	Analyst	30,000.00	90-04-01
0306	Programmer	25,000.00	90-04-01
0100	Manager	50,000.00	90-05-01
0305	Analyst	30,000.00	90-05-01
0306	Programmer	25,000.00	90-05-01

FIGURE 3.17 HISTORICAL RELATIONAL DATA MODEL (Clifford and Croker)

Also considered by Clifford and Croker in modelling the lifespan of the entity, is the lifespan of the associated database schema which defines the entity. Schemas can evolve over time and this must be

recorded and incorporated into the modelling of the entity lifespan. To handle this, attributes in the schema are timestamped. Thus, in order to find the value of an attribute in a tuple, the DBMS must check if the tuple existed at that time in addition to determining if the attribute existed in the schema at that point in time.

Clifford and Croker's temporal data model supports only a single dimension of time, the valid time, and thus represents a historical database. This data model though, also provides an approach to supporting time varying schemas in addition to supporting temporal data.

(E) Rowe and Stonebraker

Rowe and Stonebraker [33] and Stonebraker [44] have discussed in detail the POSTGRES data model and its underlying storage system. The POSTGRES DBMS has been designed with the primary purpose of extending the relational model to support abstract data types. An additional feature incorporated into the system is the retention of historic data by the DBMS. The POSTGRES query language has also been extended to support queries on the historic data.

The POSTGRES data model maintains historic data by timestamping records (tuples) with a transaction time interval. This is done by adding the attributes TMIN and TMAX to indicate when the record was inserted and when it was deleted, respectively (Figure 3.18). These timestamp attributes are transparent to the application program and are manipulated only by POSTGRES. No retroactive or proactive updates are allowed on the historic data.

Since only a single time dimension, the transaction time, is supported, the POSTGRES data model supports rollback relations only.

EMPNO	POSITION	SALARY	TMIN	TMAX
0100	Manager	50,000.00	90-01-01	-
0300	Analyst	29,000.00	90-01-01	-
0305	Programmer	26,000.00	90-01-01	-
0305	Analyst	33,000.00	90-02-01	90-03-01
0305	Analyst	30,000.00	90-03-01	-
0306	Programmer	25,000.00	90-01-01	90-05-01
0307	Programmer	25,000.00	90-04-01	-

FIGURE 3.18 POSTGRES DATA MODEL (Rowe and Stonebraker)

Summary

The majority of research on the development of a temporal data model has modified the relational data model to support timestamped tuples, as described in the previous five proprosals. This is most likely because first normal form relations are maintained, which in turn simplifies the extension of the query and manipulation languages already based on first normal form relations. Of the temporal data models based on timestamped tuples, only one (Ariav [7]) supports temporal relations using both valid and transaction times. The other temporal data models support either a single valid or transaction timestamp, and one does not specify the nature of timestamp (Clifford and Warren [12]). Since these data models are based on timestamp manipulation by the DBMS, which require modifications to the DBMS, only one of the models, POSTGRES, has actually been implemented. (Rowe and Stonebraker [33]).

3.5.3 The Relational Data Model With Non-First Normal Form Relations

The extension of the relational data model to support non-first normal form relations (including nested relations) is a third approach which has been used in the development a temporal data model. The support of non-first normal form relations permits attribute versioning to be used. This approach to developing a temporal data model has been taken by (a) Gadia, (b) Tansel and (c) Tansel and Garnett.

(A) Gadia

Gadia [23,24] states that the definition of a tuple, the basic unit of manipulation in a relational database system, should be as flexible as possible so that a tuple and its history can be manipulated as a single unit. Therefore, rather than allowing a single value for each attribute of a tuple, as in the standard relational model, Gadia has proposed that an attribute be assigned a set of values which represent the history of assigned values. In order to support such a model, Gadia has developed a temporal data model based on non-first normal form relations.

In Gadia's data model, a "temporal relation" is a collection of "temporal tuples", where each tuple is a collection of attributes called "temporal elements". Each "temporal element" is a set of data value/time interval pairs. The resulting relation is in non-first normal form (Figure 3.19a).

Gadia has defined the "temporal domain" of an attribute as the union of all the time intervals of the value/time interval pairs for the attribute. The restriction placed on Gadia's present definition of his temporal data model is that the temporal domains of all the attributes in a tuple must be the same. The tuple is said to be "homogenous" if this property is true. The temporal domains can differ from tuple to tuple, though. The homogenity restriction permits a snapshot relation, a view of the temporal relation at a particular point in time, to be extracted from the non-first normal form relation and this snapshot relation

EMPNO	POSITION		SALARY	
0100	Manager	[90-01-01: NOW]	50,000.00	[90-01-01: NOW]
0300	Analyst	[90-01-01: NOW]	29,000.00	[90-01-01: NOW]
0305	Programmer	[90-01-01: 90-02-01]	26,000.00	[90-01-01: 90-02-01]
	Analyst	[90-02-01: NOW]	30,000.00	[90-02-01: NOW]
0306	Programmer	[90-01-01: 90-05-01]	25,000.00	[90-01-01: 90-05-01]
0307	Programmer	[90-09-01: NOW]	25,000.00	[90-09-01: NOW]

FIGURE 3.19a TEMPORAL RELATION (Gadia)

will have no null valued attributes (Figure 3.19b). The homogenity restriction is placed on the initial definition of the model since the fact that no nulls are involved in the snapshots simplifies the definition of the relational operations. Gadia states that the "snapshot can be considered as the basic building block" of this model and these snapshots are in first normal form.

EMPNO	POSITION	SALARY
0100	Manager	50,000.00
0300	Analyst	29,000.00
0305	Analyst	30,000.00
0306	Programmer	25,000.00

FIGURE 3.19b SNAPSHOT AT 1990-02-15 FROM TEMPORAL RELATION (Gadia)

Not permitting null values places a severe restriction on the model and Gadia has stated that the homogenity restriction must be relaxed to support data in which the time domains for all attributes are not the same. In a temporal data model with the homogenity restriction lifted, attributes may have null values for some time intervals. This extension, Gadia states, is necessary to model real-world data and is part of the future work on this data model.

The temporal data model described by Gadia does not model temporal databases since only a single dimension of time is maintained. Gadia does not specify whether this time dimension represents transaction or valid time, so the relations can be either rollback or historical.

(B) Tansel

Tansel [45], like Gadia, has based his temporal data model on non-first normal form relations. In Tansel's data model, the attributes in a relation are timestamped with a time interval indicating the period of time in which the value is valid. Tansel argues that attribute versioning is preferred to tuple versioning since the majority of update operations affect only a subset of the attributes in a tuple, and it is only for these attributes that a new version should be created. When tuple versioning is used, even unchanged attributes are duplicated in each version of the tuple.

Tansel has chosen, like Gadia, a time interval rather that a time point as the timestamp. (Figure 3.20) This is done to simplify the relational algebra. If time points, representing the time the value became valid, were chosen, then to determine the period of validity for a value during query processing, two successive attribute values and their timestamps would have to be examined.

Attributes in Tansel's data model can be one of four types: "atomic", "set-valued", "triplet-valued" and "set-triplet-valued". Atomic and set-valued attributes are not time related. Atomic attributes are those which can contain a single value which does not vary over time. A set-valued attribute is one which can be assigned a set of atomic values. These sets of atomic values can be added to or updated but the time is not recorded. Triplet-valued and set-triplet-valued attributes are the timestamped equivalents of atomic and set-valued attributes. In these attributes, the triplet consists of the data value, either atomic or set-valued, along with

EMP NO	POSITION		SALARY		DELETE TIME
0100	Manager	[90-01-01: NOW]	50,000.00	[90-01-01: NOW]	-
0300	Analyst	[90-01-01: NOW]	29,000.00	[90-01-01: NOW]	-
0305	Programmer	[90-01-01: 90-02-01]	26,000.00	[90-01-01: 90-02-01]	-
	Analyst	[90-02-01: NOW]	30,000.00	[90-02-01: NOW]	
0306	Programmer	[90-01-01: 90-05-01]	25,000.00	[90-01-01: 90-05-01]	90-05-01
0307	Programmer	[90-09-01: NOW]	25,000.00	[90-09-01: NOW]	-

FIGURE 3.20 TEMPORAL RELATION (Tansel)

the interval start and end time. Tansel has also allowed for time varying attributes in which the timestamps of successive values can overlap. As an example, this would be the case for an employee with assignments to two projects, and one project begins before the previous one has completed.

While the update operation generally affects a subset of the attributes in a tuple, the delete operation deletes an entire tuple from a relation. To handle the delete operation, a delete time is added as an attribute of the entire tuple to indicate the time at which the tuple was deleted.

Tansel's temporal data model represents either a rollback or historical database since only a single time dimension is supported. Tansel claims that the resulting data model is flexible because different attributes types (ie. temporal and non-temporal, set valued and atomic) can co-exist in the same relation, and that the three-dimensional, non-first normal form view of the relation is maintained at the user level.

(C) Tansel and Garnett

Tansel and Garnett [47] have proposed a slight variation to the temporal data model proposed by Tansel [46]. The variations suggested are in the concept of "temporal atoms" and in the use of nested relations.

A "temporal atom" consists of a data value, atomic or set-valued, and a "temporal set", a set of non-overlapping time intervals, denoting the time periods over which the data value is valid. A temporal atom is similar to one or more of Tansel's previously defined "triplets" [46] except that common triplet values are pulled out into a single value and the time intervals grouped together. A second difference from the triplet attribute type is that there is no support for overlapping time intervals.

A second new feature of this temporal data model is that an attribute in a relation can be a temporal atom or another relation, which in turn can contain more temporal atoms or relations. Non-time varying attributes in this data model are also defined as temporal atoms with a single time interval representing the entire life of data in the relation. The history of an attribute then, is simply a collection of temporal atoms.

Thus, the temporal data model proposed by Tansel and Garnett is similar to attribute timestamping in Tansel's model [46] except that the concept of "temporal atoms" is introduced and some of the attributes in the relations can be other relations.

Summary

In summary, only three data models have proposed using timestamped attributes and non-first normal form relations to support time-varying data. The models developed by Tansel [46] and Tansel and Garnett [47] differ only in that the later replaces a triplet attribute with the more flexibly defined temporal atom and that nested attributes are permitted. In comparing Tansel's data model to the data model developed by Gadia [23,24], both have the same view of the data as a three dimensional structure based on non-first normal form relations. The first difference between these temporal data models is that Tansel has associated a delete timestamp with an entire tuple where Gadia has not. This makes it much easier to detect deleted tuples in Tansel's model, which simplifies the manipulation fo the tuples by the DBMS, since the temporal domain

of the tuple does not have to be determined. A second way in which these data models differ is in the way they manipulate the non-first normal form temporal relations. Gadia extracts snapshots, or static relations, from the non-first normal form relations and then performs operations on each snapshot, while Tansel first normalizes the non-first normal form relation, and then applies the standard relational algebra operations. The manipulation of these temporal data models will be discussed further in the following chapter. In all of the data models, only a single dimension of time is supported and the nature of the timestamp, valid or transaction time, has not been specified. None of these temporal data models has been implemented, likely since research on non-first normal form relations is still in its infancy.

3.5.4 A Temporal Specific Data Model

The development of a data model based solely on temporal data is the fourth approach to deriving a temporal data model. Shoshani and Kawagoe [40] and Segev and Shoshani [39] have taken this approach to develop a new temporal data model based on the concept of "time sequences".

Shashani and Kawagoe have approached the development of the temporal data model independent of any existing logical data models (ie. relational). The relationship between time and data was first examined at a conceptual level, and then a data model was developed to support the temporal data. After developing the temporal data model, a mapping into the relational data model was proposed. This approach to developing a temporal data model differs from the previous three in that the previous approaches have extended the relational model to support temporal data rather than developing a data model based on temporal data.

From their examination of time and data, Shoshani and Kawagoe have developed the concept of "time sequences". A "time sequence" (TS) is a collection of time/attribute value pairs for a primary key. A primary key is an attribute or attributes which uniquely identify a time sequence. For example, consider the time sequence shown in Figure 3.21a representing the salary progression for an employee. In this case, the

primary key attribute is the EMPNO, and SALARY is the temporal attribue. All like time sequences can be combined together to form a "time sequence array" (TSA) or "time sequence collection". This can be depicted conceptually as in Figure 3.21b, where each row in the matrix is a time sequence.

EMPNO	SALARY	TRANSACTION TIME
0305	26,000.00	1990-01-01
	33,000.00	1990-02-01
	30,000.00	1990-03-01

FIGURE 3.21a TIME SEQUENCE

		TRANSACTION TIME				
		1990-01-01	1990-02-01	1990-03-01	1990-04-01	....
	0100	50,000.00				
	0605	21,000.00				
	0200	30,000.00				
EMPNO	0300	29,000.00				
	0305	26,000.00	33,000.00	30,000.00		
	0306	25,000.00				
	0307				25,000.00	

FIGURE 3.21b TIME SEQUENCE COLLECTION

Whether or not a value exists in the TSA for a primary key at a given point in time depends on the properties of the data modelled by the time sequence. For example, the time sequences in Figure 3.21b are step-wise constant, which implies that the salary of an employee remains constant between points where

the data is recorded. By examining time sequence data in terms of the properties of regularity, type, static/dynamic and the time unit, a uniform way of modelling, manipulating and physically storing temporal data can be developed.

The time sequence and time sequence array are the data structures of Shoshani and Kawagoe's temporal data model. As will be seen in the chapter on temporal data manuipulation languages, a new set of operations have been developed that manipute entire time sequences and time sequence arrays. This approach has also resulted in the development of efficient storage structures and access methods based on the type of time sequence array being stored.

The example presented in Figure 3.21 is a simple time sequence collection with a single primary key attribute (EMPNO), time value (transaction time), and temporal attribute value (SALARY). Segev and Shoshani [39] also propose complex time sequence collections with composite primary keys (ie. multiple attributes form the primary key), multiple time values or multiple timestamp attributes. A time sequence collection with a composite primary key can be used to model the relationship between two entities. In the example data model, an EMPDEPT relation with the primary key attributes, EMPNO and DEPTNO, could have been used to relate employees to departments. A time sequence collection with multiple timestamp values is one which records more that one dimension of time for an attribute. Such a collection may be recording valid and transaction time, making it a temporal database. A time sequence collection with multiple temporal attributes is used to record the values of several temporal attributes at the same point in time. As an example, it may have been defined in the EMP relation, that the SALARY and POSITION attributes will always change at the same time and their values should be recorded together.

Thus, Shoshani and Kawagoe, and Segev and Shoshani have developed a temporal data model which is not based on the relational or any other data model, but rather on the concept of time sequence collections. Operators have been developed to manipulate entire time sequences and time sequence collections. Since the data model was developed by first examining the time data relationship rather than attempting to base it on

an existing model, it has the flexibility to support temporal data with valid and/or transaction time dimensions. Current research with this model involves mapping it into an existing data model, if possible, and extending the operators to handle composite primary keys, multiple timestamp values and multiple temporal attributes.

In terms of mapping the model to an existing data model, Segev and Shoshani have proposed using the relational model. The proposal suggests using a relation with three columns to represent a time sequence collection. This relation would have one column for each primary key, time and attribute value. The time sequence collection shown in Figure 3.21b could be mapped into a relation with the columns as in Figure 3.21a, having three rows for employee number 0305. This mapping to the relational model has been used by Segev and Gunadhi [38] in their discussion of the event-join operation and optimization in temporal databases. This mapping may not be ideal though, since all properties (ie. type, regularity, lifespan) of the time sequence collection are not easily represented, but must be maintained in order to perform interpolation and manipulation of the data. Thus, a special "temporal relation" may have to be developed to support this.

3.6 SUMMARY

In this chapter, several proposals for a temporal data model have been examined and many important concepts and properties of temporal data introduced. Four basic approaches have been proposed for developing a temporal data model. Those approaches are a) to build a temporal data model "on top" of the relational model, b) to modify the relational model to support tuple versioning, c) to modify the relational model to support attribute versioning, and d) to build a temporal specific data model. Collectively, the data models have introduced many features to be included in a temporal data model.

Building a temporal data model "on top" of the relational model requires no modification of the supporting DBMS. The only modifications required are the addition of timestamp attributes to each relation and of an

interface program to translate the temporal queries into queries against the underlying relations. This approach is not truly developing a temporal data model but simply an application of the relation data model to support temporal data.

The approaches based on tuple and attribute timestamping have actually proposed modifications to the relational data model to support temporal data. In the relational data model, the tuple is the basic unit of manipulation. Thus, to support temporal data, successive versions of the tuple must be maintained or the definition of the tuple modified to maintain successive versions of the temporal attributes. Temporal data models based on both have been proposed, though the use of tuple versioning is more common [12,7,35,36,37,11,33]. This is likely because the resulting temporal data model remains in first normal form, which simplifies the extension of the data manipulation language already based on first normal form relations. The temporal data models supporting attribute versioning [23,24,46] are based on non-first normal form relations, which are not supported in the relational data model. The non-first normal form temporal data models are equal in terms of flexibility and more closely model the characteristics of time varying attributes.

The temporal specific data model [39,40] based on time sequence collections is the best temporal data model approach since it was developed to specifically support this type of data, while the other approaches attempt to fit temporal data into an existing model. For this reason though, this temporal data model may be the most difficult to implement since it is not built on an existing DBMS. A proposal for mapping this model into the relational data model has been proposed, but this does not capture all the meaning associated with the time sequence collections.

The dimensions of time supported in the data model are quite important. The transaction time is not updatable by the user since it reflects the commit time of an operation on the data. Though not as important to the user, the transaction time dimension is important to the database administrator as a recovery point for the data. The valid time, though, is the more important time dimension to the user since

it represents the time the data became valid in the "real world". Valid timestamps can also be modified by retroactive and proactive updates. The data model proposed by Ariav [7] even allows the association of multiple valid time dimensions with a tuple, even though it is not apparent that this would be very useful. Most temporal data models support valid, or valid and transaction times rather than strictly transaction time.

The timestamps used in the temporal data models represent time instants or time intervals. A time interval is preferred since it allows a query to determine the entire period of validity of a temporal version without having to examine successive versions. The disadvantage of time intervals though, is that an update to the time interval is required after it has been created in order to update the end time. This complicates the use of write-once storage device for historic data. When attribute versioning is used, the temporal domain of the tuple is important since it is the union of the periods of validity of the individual attributes. Since a delete operation affects the entire tuple, in Tansel's [46] temporal data model, a delete time is associated with the entire tuple to easily identify deleted tuples.

A final important feature of a temporal data model is the ability to support time-varying schemas. Invariably, user data requirements will change over time and modifications to the schema will be required. Two approaches have been proposed to support this. The first is to make the database catalog a temporal database itself to maintain versions of the schema [14,2,11]. Queries on the data then must not only determine if the tuple was valid, but also if the attribute existed in the schema at that time. A second approach [30], is to create a new relation when a schema is modified and then have the query language translate temporal queries into queries on the current and prior relations and combine the tuples from each to give the final result.

A temporal data model consists not only of the data structures as described in this chapter, but also the data language required to access and manipulate the data structures. The temporal data language will be discussed in the Chapter 4 and Chapter 5 will cover the physical implementation of the proposed temporal data structures.

4. TEMPORAL DATA LANGUAGE

4.1 INTRODUCTION

The second part of a data model is the set of operators to define and manipulate the data structures of the model. The data language component of the temporal data model must also be enhanced to support temporal data. A data language supporting temporal data will be referred to as a temporal data language. The data language component of a data model is comprised of the data defintion language, the data manipulation language and the data control language. The data control language, used to authorize access to the data, has not been enhanced in any of the research reviewed and will not be discussed in this chapter.

In general, a temporal data language should a) allow a relation to be defined as a snapshot, rollback, historical or temporal relation, b) allow the definition of constant and time varying attributes within a particular relation if selective versioning of attributes is permitted, and c) allow for the querying and manipulation of the temporal data over the time dimension.

Just as there are several approaches to developing a temporal data model, there are several varied approaches to extending the relational operators and languages to support temporal data. The intent of this chapter is to outline the enhancements to the higher level query languages (ie. SQL) where possible, and in models where such extensions are not presented, discuss enhancements at the relational algebra level. The formal semantics of the languages and the formal relational algebra and calculus definitions to support the enhanced relational operators will not be described in detail.

The discussion of temporal data languages is organized as follows. Section 4.2 presents the data definition

language enhancements to permit the definition of snapshot, rollback, historical and temporal relations. Temporal data manipulation languages are covered in section 4.3. This discussion includes temporal enhancements for relational algebra and existing relational languages (ie. QUEL, SQL), and an introduction to a new temporal data manipulation language for time sequences and time sequence collections. When a temporal query is made against an attribute storing continuous data, the value of the attributes at a time not explicitly recorded in the database may be requested. In these situations, a function is required to determine this value. These functions are called derivation and approximation functions and are introduced in Section 4.4.

4.2 TEMPORAL DATA DEFINITION LANGUAGE

The literature on temporal data languages concentrates primarily on data manipulation language enhancements, rather than the relatively straightforward extensions to the data definition langage (DDL). In order to support temporal relations, the DDL must specify whether the relation is a snapshot, rollback, historical or temporal relation and allow for selective versioning of attributes if it is supported in the model.

Snodgrass [42] has outlined DDL extensions to the QUEL CREATE statement to permit the definition of the different temporal relation types. The DDL in Figure 4.1 defines the sample EMP relation as a temporal relation. The keyword PERSISTENT is specifed on the CREATE statement to add the "transaction time" dimension to the relation when defining a rollback or temporal relation. The keywords INTERVAL or EVENT are specified on the CREATE to add the valid time dimension to relations, either as an interval or as a time instant. These keywords are used when defining historical or temporal relations. Note that the transaction time is always specified as a time interval in order to indicate when the tuple is inserted and deleted. The valid time, on the other hand, can be a time interval or time instant, depending on

the nature of the data being modelled.

```
CREATE PERSISTENT INTERVAL EMP( EMPNO = I4
                                EMPNAME = C20
                                POSITION = C10
                                SALARY = P7.2
                                DEPTNO = C3
                                BIRTHDATE = C10)
```

FIGURE 4.1 DDL TO DEFINE TEMPORAL RELATION "EMP" (Snodgrass)

Sarda [36,37] has extended the SQL CREATE statement to define the historical relations based on timestamped tuples permitted in his data model (Figure 4.2). The keyword STATE is used to specify a historical relation with interval timestamps. The keyword EVENT could be specified in place of STATE to record event data using instant timestamps. The WITH TIME clause is used to specify the granularity of the timestamps.

```
CREATE STATE TABLE EMP ( EMPNO       INTEGER
                         EMPNAME     CHAR(20)
                         POSITION    CHAR(10)
                         SALARY      DECIMAL(7,2)
                         DEPTNO      CHAR(3)
                         BIRTHDATE   CHAR(10) )
    WITH TIME GRANULARITY OF YEAR:MONTH:DAY
```

FIGURE 4.2 DDL TO DEFINE HISTORICAL RELATION "EMP" (Sarda)

Both of these versions of temporal DDL support temporal data models based on tuple versioning. No examples of selective attribute versioning are available since no temporal DDL has been presented for the temporal data models based on attribute versioning.

4.3 TEMPORAL DATA MANIPULATION LANGUAGE

The data manipulation language can be considered to have two components, a query component and an update component. All research regarding extensions to the data manipulation language to support temporal data, covers extensions to the query component to a greater extent than extensions to the update component. This is likely because the number of variations of temporal updates are much fewer than those of temporal queries, in which many variations are possible. For this reason, this chapter will focus primarily on temporal queries, and discuss temporal updates only where they have been proposed.

There are two general types of queries over the time dimension. The first retrieves a tuple or attribute value at a particular point in time. For example, "what was Jones' salary at 1990-02-15?". The second manipulates a sequence of values for a tuple or attribute over time. For example, "what was Jones' salary history during 1990". These have been termed "as of" and "walk through time" queries (Dadum, Lum, Werner [14]). Using the image of a time oriented relation with a single time dimension as a three-dimensional cube with time as the third dimension, "as of" queries manipulate a single vertical "slice" of a relation where "walk-through-time" queries manipulate a sequence of vertical slices.

The following sections of this chapter will outline the proposed enhancements to the relational operations and query languages to support access to time oriented data, grouped according to the following:

(1) enhancements to the basic relational algebra operations

(2) enhancements to higher level relational languages such as QUEL and SQL

(3) new temporal data manipulation languages

4.3.1 Enhanced Relational Algebras

The first approach to providing access to temporal data involves the enhancement of the basic relational algebra operations (ie. SELECT, PROJECT and JOIN).

The standard SELECT operator selects a horizontal subset of a snapshot relation, a group of tuples, based on attribute values within the tuples. The standard PROJECT operator produces a vertical subset of a snapshot relation by selecting only certain attributes of a relation from all tuples. Temporal enhancements at this level have extended the SELECT and PROJECT operators to function over a time dimension, in addition to adding entirely new operations to manipulate the time dimension of the data. The standard JOIN operator is used to combine two relations over a common attribute. The JOIN operator in the temporal data model has been extended to join tuples over both the attribute and time dimensions in the same operation.

The basic relational algebra operators, SELECT, PROJECT and JOIN, form the basis of some of the higher level data manipulation languages such as SQL. Extensions at the relational algebra level have been proposed by (a) Clifford and Croker, and (b) Gadia and Tansel.

(A) Clifford and Croker

Clifford and Croker [11] have proposed query language extensions to support temporal data only at the relational algebra level. This has been done by augmenting the basic operations of SELECT and JOIN, in addition to adding new operations, such as TIME-SLICE and WHEN, which operate solely in the time dimension.

EMPNO	POSITION	SALARY	TIMESTAMP
0305	Programmer	26,000.00	90-01-01
0305	Analyst	30,000.00	90-02-01
0305	Analyst	30,000.00	90-03-01
0305	Analyst	30,000.00	90-04-01
0305	Analyst	30,000.00	90-05-01

FIGURE 4.3a SELECT-IF(EMPN=0305 AND POSITION=PROGRAMMER)

The SELECT operation has been extended to a SELECT-IF and a SELECT-WHEN. The SELECT-IF operation selects tuples from a relation where the selection criteria is matched for an attribute at any point in time, and produces a result relation comprising the selected tuples and their entire lifespan. In Figure 4.3a, a SELECT-IF is performed on Clifford and Croker's historical relation from Chapter 3 (Figure 3.17). The SELECT-WHEN operation also selects tuples where the selection criteria matches the attribute values at any point in time, but the resulting relation contains the selected tuples with new lifespans which include only the points in time where the selection criteria was met (Figure 4.3b). The SELECT-WHEN operation "reduces" the relation not only in the attribute value dimension, as is done with standard SELECT, but also along the time dimension.

EMPNO	POSITION	SALARY	TIMESTAMP
0305	Programmer	26,000.00	90-01-01

FIGURE 4.3b SELECT-WHEN(EMPN=0305 AND POSITION=PROGRAMMER)

The TIME-SLICE operation is used to reduce a relation along the time dimension. This operation returns a relation containing the tuples which have a lifespan containing the time points specified, or corresponding

to the lifespan of an attribute in the relation (Figure 4.3c).

EMPNO	POSITION	SALARY	TIMESTAMP
0100	Manager	50,000.00	90-01-15
0305	Programmer	26,000.00	90-01-15
0306	Programmer	25,000.00	90-01-15
0100	Manager	50,000.00	90-02-01
0305	Analyst	30,000.00	90-02-01
0306	Programmer	25,000.00	90-02-01

FIGURE 4.3c TIME-SLICE(90-01-15,90-02-01)

The WHEN operation returns a set of times indicating when a selected set of tuples is defined. This operation can be used in conjunction with the SELECT and TIME-SLICE operations to provide a set of times to be used as input selection criteria for other operations. For example, a WHEN(EMPNO=0305 and POSITION=PROGRAMMER) would return the lifespan [90-01-01,90-01-01]. The data associated with this lifespan is not returned.

The standard JOIN operation is enhanced to join tuples where the lifespans intersect as well as the selection criteria is matched. A new TIME-JOIN operation is also defined in which one relation is joined with another relation using a standard JOIN, and the resulting rows selected from the composite relation are those which have a lifespan intersecting with the lifespan of a time varying attribute in the first relation. Essentially, a TIME-SLICE is performed on the the first relation before it is joined, using a standard JOIN, to the second.

(B) Gadia and Tansel

Gadia [23,24] and Tansel [46] have both developed temporal data models based on non-first normal form

relations with timestamped attributes supporting a single dimension of time. The underlying structure of the relations in both approaches is very similar, but the way in which the data is accessed is different. For this reason, their relational algebra enhancements will be discussed together in this section.

In general, the manipulation of the temporal relations by Gadia and Tansel are similar in that they both, ultimately, manipulate normalized relations. They differ though, in their approach to normalization. Recall that the basic building block of Gadia's temporal data model is a snapshot. When Gadia accesses a temporal relation, a snapshot, or set of snapshots, in first normal form, are extracted from the non-first normal form temporal relation, and the relational algebra operations are applied to each snapshot. In Tansel's approach, the entire non-first normal form temporal relation is first normalized, and then the relational algebra operation is applied. Neither Tansel nor Gadia have provided a higher level query language extension, such as SQL or QUEL, to perform these operations, but have defined the extensions to the standard relational algebra operations.

In Tansel's approach, additional operators are defined to normalize the non-first normal form relations before applying the standard relational algebra operations. The additional operations defined are PACK, UNPACK, T-DEC, T-FORM, SLICE and DROP-TIME.

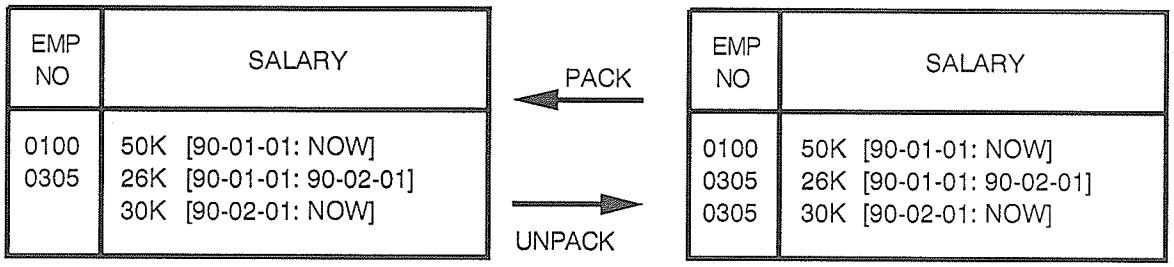


FIGURE 4.4a PACK and UNPACK OPERATIONS (Tansel)

The UNPACK and PACK operations work with set-valued attributes. The UNPACK operation converts a tuple with a set-valued attribute into multiple tuples, each containing one element from the set for that

attribute. Applying the UNPACK operation to each set valued attribute in a relation will convert the relation into first normal form (Figure 4.4a). The PACK operation performs the opposite operation, grouping multiple tuples over an attribute into a single tuple with a set-valued attribute.

The T-DEC and T-FORM operations work with triplet-valued (ie. timestamped) attributes. The T-DEC operator performs triplet decomposition, breaking a triplet-valued attribute into three attributes containing the attribute value, the start time and the stop time (Figure 4.4b). The T-FORM operation creates a single triplet-valued attribute from a value, a start time and a stop time attribute.

EMP NO	SALARY
0100	50K [90-01-01: NOW]
0305	26K [90-01-01: 90-02-01]
0305	30K [90-02-01: NOW]

T-FORM

T-DEC

EMP NO	SALARY	SALARY START	SALARY END
0100	50K	90-01-01	NOW
0305	26K	90-01-01	90-02-01
0305	30K	90-02-01	NOW

FIGURE 4.4b T-DEC and T-FORM OPERATIONS (Tansel)

The SLICE operator works on relations with more than one temporal attribute by splitting the triplets of an attribute according to the time intervals of another temporal attribute. This operation takes the intersection of the time intervals of the two attributes (Figure 4.4c). Two other variations on this operation, USLICE and DSLICE, have also been proposed based on the union and difference operations on the attributes time intervals.

The DROP-TIME operation simply drops the time component of a triplet-valued or set-triplet-valued attribute to produce atomic or set-valued attributes, respectively.

EMP NO	SALARY	MANAGER
0305	26K [90-01-01: 90-02-01] 30K [90-02-01: NOW]	0100 [90-01-01: 90-01-15] 0200 [90-01-15: 90-03-01] 0300 [90-03-01: NOW]

SLICE (SALARY,MANAGER)

EMP NO	SALARY	MANAGER
0305	26K [90-01-01: 90-02-01]	0100 [90-01-01: 90-01-15] 0200 [90-01-15: 90-02-01]
0305	30K [90-02-01: NOW]	0200 [90-02-01: 90-03-01] 0300 [90-03-01: NOW]

FIGURE 4.4c SLICE OPERATION (Tansel)

The above operations are used to normalize a non-first normal form relation before applying the standard relational algebra operations.

Gadia's [23,24] data manipulation language, on the other hand, is based on snapshots. A snapshot is a static "two dimensional" relation. A static snapshot can be extracted from a temporal relation for any time point within the temporal domain of a relation. Thus, in order to perform an operation on a temporal relation, the standard relational operation is simply performed on all of the individual snapshots making up the temporal relation.

Gadia has defined the new operator, TDOM, to extract a temporal domain from a temporal relation which can then be used in subsequent operations. This operator is used only as part of another operation, such as a SELECT, to select a set of snapshots from the temporal relation to be manipulated.

4.3.2 Enhanced Data Manipulation Languages

The second category of temporal language enhancements are those made to a higher level query language such as QUEL or SQL. Two methods of extending such query languages have been proposed. The first is to add new clauses to the language to handle the time dimension separate from the clauses which manipulate the attribute dimension. This has been done by (a) Snodgrass, (b) Ariav, (c) Bassiouni and (d) Sarda. The second method is to add new temporal operators or functions to be used within existing clauses in the language to manipulate data in the time dimension. This approach has been used by (d) Sarda and (e) Adiba and Quang. Sarda has combined both approaches in his temporal query language.

To aid in comparison of the languages, the following temporal query against the sample EMP relation from Chapter 3 (Figure 3.10) will be used for all of the approaches: "What was Jones' salary on 1990-02-15?" This "as of" query can have different responses depending on the temporal data model that is used to implement the EMP relation. If the EMP relation is a rollback relation, then the result is $33,000. This is because in a rollback relation, the retroactive update is treated as a real time update and the past data is not modified. If the EMP relation is a historical relation, then the result is $30,000. In this case, the data is timestamped with a valid timestamp and the retroactive update corrects the salary at 1990-02-01 to $30,000. If the EMP relation is a temporal relation, then the query can be written to retrieve either of these results.

(A) Snodgrass

Snodgrass [42] has developed an extension of the INGRES query language QUEL, called TQUEL, to support temporal relations. The general format of a QUEL query statement is:

RANGE OF <source relation>;

RETRIEVE INTO <target relation>

WHERE <object selection qualification> ;

In order to support the additional time dimensions of a temporal relation, clauses have been added to the RETRIEVE statement resulting in a query of the form:

RANGE OF <source relation>;

RETRIEVE INTO <target relation>

WHERE <object selection qualification>

WHEN . . .

VALID FROM . . . TO . . .

AS OF . . .;

The WHEN clause is used to restrict time values in the valid time dimension. It allows the query to select data which is valid at a particular "real world" time. It is analogous to the WHERE clause for attribute values. It is also possible to specify a VALID clause on a RETRIEVE statement to indicate the valid times to be associated with the values in the target relation.

To manipulate time in the "transaction time" dimension, the AS OF clause is used. The AS OF clause causes the query to be performed on the state of the relation as it was at a specified time. If no AS OF clause is specified, AS OF the current time is assumed and no roll back to a prior state is done. The AS OF clause can also be used to do a "walk through time" query by specifying the THROUGH keyword on the AS OF clause, as in AS OF ... THROUGH ... .

The temporal data model developed by Snodgrass represents a temporal database supporting valid and transaction time. The example query can be answered with the queries in Figure 4.5. In Figure 4.5a, a roll

back is done to the state of the relation at 1990-02-15 and the result of $33,000 is returned. In Figure 4.4b, no rollback is performed and the result is $30,000. The OVERLAP keyword indicates the intersection of the valid time associated with the tuple and the required valid time, 1990-02-15.

```
RANGE OF E IS EMP
RETRIEVE INTO RESLTS (SAL = E.SALARY)
    WHERE E.EMPNAME = 'JONES'
    WHEN E OVERLAP '1990-02-15'
    AS OF '1990-02-15'
```

(a)

```
RANGE OF E IS EMP
RETRIEVE INTO RESLTS (SAL= E.SALARY)
    WHERE E.EMPNAME = 'JONES'
    WHEN E OVERLAP '1990-02-15'
```

(b)

FIGURE 4.5 TEMPORAL QUERY LANGUAGE TQUEL (Snodgrass)

Both the WHEN and VALID clauses can be used with the REPLACE statement when doing updates to historical data based on valid time. The WHEN clause can be used to select the particular tuples to be updated, and the VALID clause can be used to specify new valid time values. The AS OF clause cannot be used in the REPLACE since the transaction time is computed and added automatically by the system.

(B) Ariav

Ariav [7] has developed a "temporally oriented" version of SQL, TOSQL, which extends the language to function over the temporal domain. In the temporal data model developed by Ariav, a relation has at least one time dimension, the transaction time (called the recording time by Ariav), and can optionally have a valid time dimension.

Ariav has extended the SQL SELECT to support temporal relations by adding a time specification (AT,WHILE,...) and time qualification (AS-OF) clause. The general format of the extended SELECT

statement is:

SELECT <object selection>
FROM <source relation>
WHERE <object qualification>
AS-OF <time qualification> ALONG <time dimension>
[AT|WHILE|...] <time period spec> ALONG <time dimension>

The time qualification in the AS-OF clause specifies the time along a time dimension the query is to be run. This causes the relation to be rolled back to a prior state. The AS-OF clause specifies both a time and time dimension along which the rollback operation is to be done.

The selection criteria for data associated with time values along a particular time dimension is done with the time period specification clause. This clause uses the new operators of AT, WHILE, DURING, BEFORE or AFTER followed by the time period in which the data is to exist. The time dimension is specified in the ALONG clause.

Ariav's temporal data model supports temporal databases with a single transaction time dimension and one or more valid time dimensions. In TOSQL, at most two time dimensions can be referenced in a single query. It is unlikely that more than two would be required in a single query since generally, only the transaction time and a single valid time time are used. As mentioned in Section 3.5.2 of the previous chapter, the usefulness of multiple timestamps for the valid time dimension is limited.

The example query is solved using TOSQL in Figure 4.6. Assuming EMP is a temporal relation, Figure 4.6a does not do a rollback using the transaction time to the relation as it existed at 1990-02-15 (ie. AS-OF PRESENT ALONG TRANSACTION-TIME) and returns the value known as of the present time, $30,000.

Figure 4.6b does do a rollback to the state at 1990-02-15 (ie. AS-OF '1990-02-15' ALONG TRANSACTION-TIME) and returns the value as it was known on 1990-02-15, $33,000.

```
SELECT EMPNAME, SALARY
   FROM EMP
   WHERE EMPNAME = 'JONES'
   AS-OF PRESENT ALONG TRANSACTION-TIME
   AT '1990-02-15'  ALONG  VALID-TIME
                  (a)

SELECT EMPNAME, SALARY
   FROM EMP
   WHERE EMPNAME = 'JONES'
   AS-OF '1990-02-15' ALONG TRANSACTION-TIME
   AT '1990-02-15'  ALONG  VALID-TIME
                  (b)
```

FIGURE 4.6 TEMPORAL QUERY LANGUAGE (Ariav)

The results of the TOSQL SELECT can be a snapshot relation (called a "standard" query), a view of a tuple over time (a "time slice" query) or a sub-cube of the cubic relation (a "cubic" query) depending on the use of the time specification and qualification clauses. Ariav also proposes that "aggregation functions", such as COUNT and SUM, be used to reduce the result relation along the attribute dimension, or along the time dimensions. For example, to count, for each current primary key, the number of occurrences of a particular attribute value over a time. This is an example of a "walk through time" query.

(C) Bassiouni

The temporal query language extension proposed by Bassiouni [8] is based on an enhanced relational data model using attributes timestamped with a single valid or transaction timestamp, as has been proposed by

Gadia and Tansel.

Bassiouni's query language is unique in that it separates the clause which selects the tuples from the one that selects the temporal values from the tuple. This syntax change is intended to simplify the usage of the statement rather than to provide additional functionality. The QUEL-like language extends the RETRIEVE statement to use a WHERE clause to select the desired tuples and a SUBJECT TO clause to select the versions of the temporal attributes to place in the result relation. This results in a statement of the form:

RANGE OF <source>;
RETRIEVE INTO <target>
SUBJECT TO <output qualification>
WHERE <object selection qualification> ;

The feature of this temporal data language which is intended to provide added flexibility in developing temporal queries, is the extension of the boolean and comparison operators so that there now exists a logical and a temporal version of each. For example, there is a logical and temporal version of the AND (AND_L and AND_T) and = ($=_L$ and $=_T$) operators.

The logical and temporal versions of the boolean and comparison operators function in a similar manner. The temporal operators compare attribute values that occur at the same time (ie. overlapping time intervals) while the logical operators compare attribute values regardless of the time they occur. The temporal versions of the operators return a set of time intervals of when the expression is evaluated to be true. The logical versions of the operators simply return an interval <0,NOW> for true and the null interval for false.

Expressions using these operators are used in the SUBJECT TO and WHERE clauses of the extended RETRIEVE statement to produce the result relation. The example query can be handled in this query

language as shown in Figure 4.7. In this example, the WHERE clause specifies that the tuples where the EMPNAME is 'JONES' are to be selected and the SUBJECT TO clause is used to select the SALARY of these employees at '1990-02-15' for the target relation.

```
RANGE OF E IS EMP
RETRIEVE (E.EMPNAME, E.SALARY)
   SUBJECT TO E.SALARY AND_T <'1990-02-15','1990-02-15'>
   WHERE E.EMPNAME =_T 'JONES'
```

FIGURE 4.7 HISTORICAL QUERY LANGUAGE (Bassiouni)

In addition, several unary operators are defined which work with sets of time intervals produced by the boolean and comparison expressions. These include WHEN, VALUE, CURRENT, START, FINISH, and FIRST_START. As well, several operators to compare intervals, such as STARTS_BEFORE, PRECEDES, SUBSET_OF and COVERED_BY, are defined.

(D) Sarda

Sarda [35,36,37] has proposed an enhanced version of SQL, called HSQL, to support queries on his relational data model which has been extended to support rollback databases.

The SQL query language is extended by adding new operations and functions to be used within existing clauses, such as the WHERE and GROUP BY clauses, and by adding the new clauses, such as FROMTIME and EXPAND. New functions and operations to manipulate the time dimension include functions such as DATE, MONTH and YEAR, and time interval operations such as concatenation, inclusion, MEET, OVERLAP.

The basic SELECT statement in HSQL is as follows:

FROMTIME <time1> TOTIME <time2>

SELECT <column selection>

FROM <source relation>

WHERE <object qualification>

EXPAND BY .....

GROUP BY .....

HAVING .....;

The use of the clauses in the HSQL SELECT statement are as follows:

a) the FROMTIME clause is used to specify a time-slice of the historical data from time1 to time2

b) the SELECT clause indicates the columns to be selected from the source relations specified in the FROM clause.

c) the FROM clause, in which the source relation is specified. By default the current relation and all its history is implied. Optionally, HISTORY(R) or CURRENT(R) can be used in the FROM clause to restrict access to only the history or current version of the relations.

d) the WHERE clause specifies object qualfications as in standard SQL in addition to allowing qualifications using the time attributes.

e) the EXPAND clause is used to convert data associated with time intervals to time instant data of a specified granularity.

f) the GROUP BY and HAVING clauses function as in SQL, in addition to allowing the use of time attributes in the grouping.

The example query is solved as shown in Figure 4.8. Since Sarda's data model supports only rollback data, the query will give the result of $33,000.

```
FROMTIME '1990-02-15' TOTIME '1990-02-15'
SELECT EMPNAME, SALARY
   FROM EMP
   WHERE EMPNAME = 'JONES'
```

FIGURE 4.8 HISTORICAL QUERY LANGUAGE (Sarda)

Sarda claims that the relational update operations are the same as for a non-temporal relation except that the DBMS manipulates the transaction timestamp on the data. Sarda also proposes extensions to support retroactive update operations though these are not allowed in true rollback relations. Retroactive updates are written by prefixing the standard SQL update operation with a FROMTIME or AT clause to specify the transaction time at which the update was to have taken place.

(E) Adiba and Quang

Adiba and Quang [2] have developed the TIGRE data model for historical databases and have extended the SQL-like query language LAMBDA to support data access. They claim that in the manipulation of historic data, time can be specified explicitly or implicity in a query. It is explicity specified when a request is made for the value of an attribute at a time *t*. Implicit or relative time specification is done when a request is made for the last *n* versions of an attribute.

To support these two types of queries, the keyword VERSION is added to the object selection clause of the SELECT statement as in the following:

SELECT VERSION AT <time specification>

OF <object selection>

FROM <source relation>

WHERE <object qualification>

For explicit queries, the form "VERSION AT t" or "VERSION AFTER t" is used to qualify attributes retrieved. For implicit queries, "LAST VERSION" or "1..3 VERSION" is added to qualify the attributes being selected. Since this model supports historical relations only, the result of the example query is $30,000 (Figure 4.9).

```
SELECT VERSION AT '1990-02-15' OF E.EMPNAME, E.SALARY
   FROM EMP
   WHERE E.EMPNAME = 'JONES'
```

FIGURE 4.9 HISTORICAL QUERY LANGUAGE LAMBDA (Adiba and Quang)

A variation of the UPDATE operation has also been defined which is used specifically to correct (update) historic data rather than add new historic data. This is done with a new statement called CORRECT. The CORRECT statement can use the VERSION keyword as descibed above in order to select specific historic data to be updated.

4.3.3 New Data Manipulation Languages

The development of an entirely new language is the final approach to providing a temporal data manipulation language. A new data manipulation language has been proposed by Shoshani and Kawagoe [40] and Segev and Shoshani [39] to support the temporal specific data model based on time sequences and time sequence collections. This method is different from the previous approaches in that no formal

underlying relational algebra has been proposed and the data is not manipulated in terms of relations, but in terms of time sequences.

Shoshani and Kawagoe, and Segev and Shoshani, have proposed a language to support temporal data which has been modelled using time sequences and time sequence collections. Shoshani and Kawagoe have discussed these operations in general terms while Segev and Shoshani have described an SQL-like language to handle these operations.

The operations, in general, have been designed to manipulate entire time sequence collections. The operations defined are SELECTION, AGGREGATION, ACCUMULATION, RESTRICTION and COMPOSITION. The basic language construct is:

operation INTO target-tsc function
FROM source-tsc
WHERE target-specification
GROUP BY|TO mapping-specification

SELECTION is used to extract a subarray of a time sequence collection by restricting the time and attribute values in the WHERE clause. Assume a simple EMP_SALARY time sequence array as shown in Figure 3.21b in chapter 3. The query that retrieves the salary for JONES at 1990-02-15 is shown in Figure 4.10, where T is the timestamp value. The result of this query is $33,000 since it is assumed that the single time dimension supported in this simple time sequence array is the transaction time.

```
SELECT INTO EMP_JONES_SALARY   SALARY
   FROM EMP_SALARY
   WHERE EMPNO = 0305
      AND T IN ('1990-02-15')
```

FIGURE 4.10 SELECTION OPERATION (Segev and Shoshani)

The remaining operations will be explained with the use of the example time sequence collections describe below and shown in Figure 4.11:

a) AUTO_SALES, containing the number of automobiles sold per day (QUANTITY) by automobile type (TYPE), where TYPE is the primary key and each time point represents a single day.

b) AUTO_PRICE, containing the price (PRICE), which is assumed to be fixed, by automobile type (TYPE). This is a non-temporal time sequence array, represented as a single column, where TYPE is the primary key.

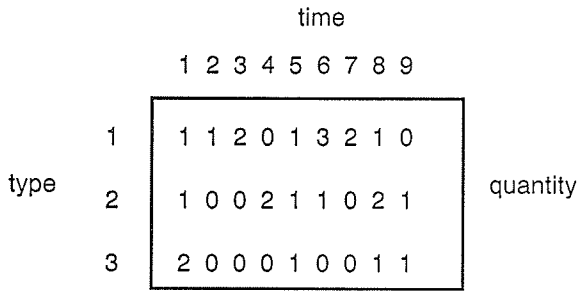
c) TAX_RATE, containing the percentage of daily revenue to be paid in taxes per day (TAX_PCT), assuming this value may vary daily. This time sequence array is a single row.

d) MFR_REBATE, containing the rebate amount (REBATE_AMT) by automobile type (TYPE) per day. Again TYPE is the primary key.

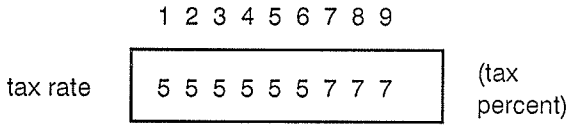
e) AUTO_COLOR, containing the color (COLOR), which is assumed to be fixed, by automobile type (TYPE). This is a non-temporal time sequence array , represented as a single column.

AGGREGATION is used to combine values over the key attributes or time dimensions. When performing aggregation over the time domain, the resulting time sequence collection has a new time unit with a higher granularity. For example, daily sales amounts may be combined to produce weekly totals as in Figure 4.11a. In this example the GROUP clause is used to group the time values by week. When aggregating values over a key attribute, the new key values of the resulting time sequence array are usually other attribute values. For example, the daily sales amounts could be combined to generate daily sales amounts by color as in Figure 4.11b. The GROUP clause is used here to indicate aggregate by key value. Note that an implicit join, by primary key, of the AUTO_SALES and AUTO_COLOR time sequence collections is performed in the query to derive the result.

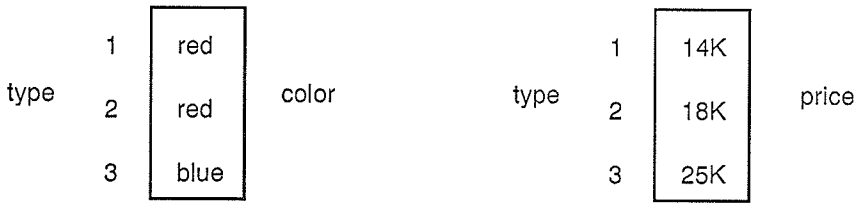
ACCUMULATION operates over the time dimension to generate new values for each time point based on the values preceeding or following it. The resulting time sequence collection has the same number of time sequences and time points, but the time sequence type may change. This is different from the



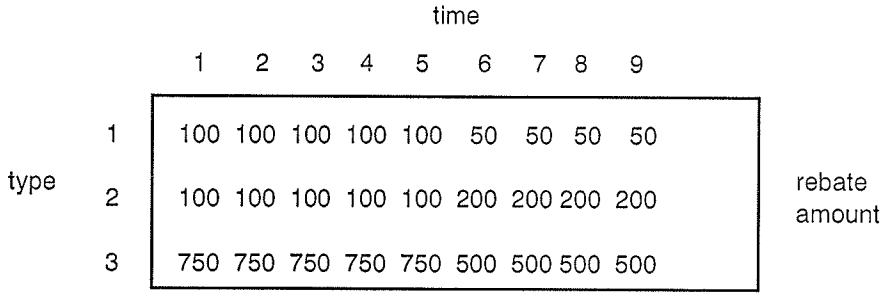
AUTO_SALES



TAX_RATE



AUTO_COLOR AUTO_PRICE



MFR_REBATE

FIGURE 4.11 SAMPLE TIME SEQUENCE COLLECTIONS

AGGREGATE INTO WEEKLY_SALES
SUM QUANTITY
FROM AUTO_SALES
GROUP T BY WEEK

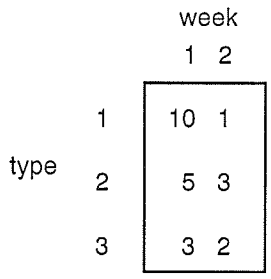


FIGURE 4.11a AGGREGATION OPERATION (Segev and Shoshani)

AGGREGATE INTO COLOR_SALES
SUM QUANTITY
FROM AUTO_SALES
GROUP K BY AUTO_COLOR COLOR

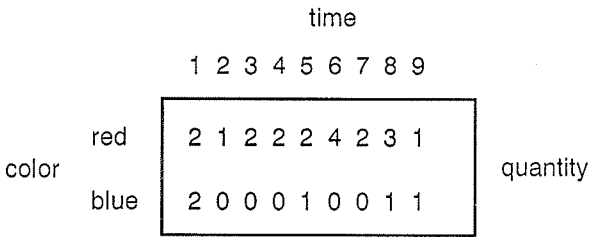


FIGURE 4.11b AGGREGATION OPERATION (Segev and Shoshani)

AGGREGATION operation which actually changes the number of time sequences or number of time points (see Figure 4.11a). As an example, ACCUMULATION may be applied to the AUTO_SALES time sequence collection, which is discrete, to produce a step-wise constant time sequence collection containing a

running total of the number sold by automobile type from the beginning of the year (Figure 4.11c). In this example, the GROUP TO BEGIN clause is used to indicate that the accumulation should start with the first time point in the time sequence collection.

```
ACCUMULATE INTO RUNNING_TOTAL_SALES
        SUM QUANTITY
    FROM AUTO_SALES
        WHERE T > '1991-01-01'
    GROUP TO BEGIN
```

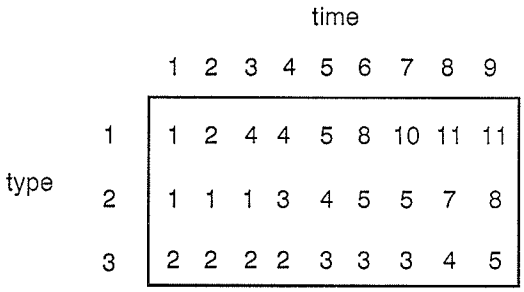


FIGURE 4.11c ACCUMULATION OPERATION (Segev and Shoshani)

RESTRICTION selects time sequences from a source time sequence collection where a time sequence with the same primary key also appears in a second time sequence collection. The selection of the source time sequences can also be restricted in the BY clause. For example, the history of the number of automobiles sold for automobiles with a price greater than $20,000 can be generated as in Figure 4.11d. In this example, an implicit join of the AUTO_PRICE and AUTO_SALES time sequence collections by primary key is done. Note that this language differs from SQL in that all tables accessed do not have to be referred to in the FROM clause.

COMPOSITION is used to combine two time sequence collections to produce a third one. There are three types of composition operations: by time; by primary key; or pairwise. COMPOSITION by time combines a time sequence collection with a single time point for each key (ie. a non-temporal time

RESTRICT INTO EXP_AUTO_SALES
FROM AUTO_SALES
BY AUTO_PRICE.PRICE > 20,000

		time								
		1	2	3	4	5	6	7	8	9
type	3	2	0	0	0	1	0	0	1	1

FIGURE 4.11d RESTRICTION OPERATION (Segev and Shoshani)

sequence collection with one attribute - a single column, such as AUTO_PRICE) with a second time sequence collection with the same key values. The result is derived by combining each column in the second time sequence collection with the values from the first. For example, AUTO_SALES could be combined with AUTO_PRICE to produce a time sequence collection, DAILY_REVENUE containing the daily revenue by automobile type (Figure 4.11e). The BY T clause, where T is the timestamp attribute, indicates that the COMPOSITION is for each time point.

COMPOSE INTO DAILY_REVENUE
REVENUE = QUANTITY * PRICE
FROM AUTO_SALES, AUTO_PRICE
BY T

		time								
		1	2	3	4	5	6	7	8	9
type	1	14K	14K	28K	0K	14K	52K	28K	14K	0K
	2	18K	0K	0K	36K	18K	18K	0K	36K	18K
	3	50K	0K	0K	0K	25K	0K	0K	25K	25K

FIGURE 4.11e COMPOSITION OPERATION (Segev and Shoshani)

For COMPOSITION by primary key, one of the source time sequence collections is a simple time sequence for a single key value (ie. a single row, such as TAX_RATE) which has the same time granularity and lifespan as a second time sequence collection. The COMPOSITION operation produces a third time sequence collection in which the simple time sequence has been combined with each time sequence in the second input time sequence collection by corresponding time points. As an example, the daily amount of tax owing by automobile type could be generated by combining the DAILY_REVENUE with the TAX_PCT as in Figure 4.11f. The BY K clause indicates that the COMPOSITION is for each primary key.

```
COMPOSE INTO TAX_OWING
        TAX = REVENUE * TAX_AMT
    FROM DAILY_REVENUE, TAX_RATE
     BY K
```

		time								
		1	2	3	4	5	6	7	8	9
type	1	0.7K	0.7K	1.4K	0K	0.7K	2.6K	1.4K	0.9K	0K
	2	0.9K	0K	0K	1.8K	0.9K	0.9K	0K	2.5K	1.2K
	3	2.5K	0K	0K	0K	1.2K	0K	0K	1.7K	1.7K

FIGURE 4.11f COMPOSITION OPERATION (Segev and Shoshani)

In pairwise COMPOSITION, each time sequence collection must have the same primary key, time granularity, and lifespan. In this operation, the corresponding values from each time sequence collection are combined to produce the new value. Pairwise COMPOSITION could be used to combine the daily sales amount with the daily manufacturers rebate amount to generate the total rebate amount by automobile by day as in Figure 4.11g. When no BY clause is specified in a COMPOSE statement, the default is pairwise composition.

COMPOSE INTO REBATES_PAID
REBATE = QUANTITY * REBATE_AMT
FROM AUTO_SALES, MFR_REBATES

		time								
		1	2	3	4	5	6	7	8	9
type	1	100	100	200	0	100	150	100	50	0
	2	100	0	0	200	100	200	0	400	200
	3	1500	0	0	0	750	0	0	500	500

FIGURE 4.11g COMPOSITION OPERATION (Segev and Shoshani)

4.4 DERIVATION AND APPROXIMATION FUNCTIONS

Queries against temporal data are often of the type "what is the value of this attribute at a given time in history". If the attribute type is constant or even step-wise constant the value can be easily determined. If, on the other hand, the attribute stores continuous data and a query requests an attribute's value at a time not explicitly recorded in the database, the derivation or approximation of this value is required.

Klopprogge and Lockemann [26] have disscussed the issue of uncertainty and have defined the concepts of derivation and approximation. "Derivation" is the process of determining with certainty the value of an attribute at a given time. For example, if the data followed a distinct formula, then the value at a certain time can be derived with certainty using a "derivation function". If there is uncertainty in computing the value, then an "approximation" is required to determine the value. This uncertainty may arise with continuous data where data does not follow a known formula or the recorded values on which the derivation

is based are not accurate due to estimates or physical measurements. The approximation is performed with a function, called an "approximation function", which may use linear interpolation or the least squares method depending on the nature of the data to approximate the required value.

Klopprogge and Lockemann have proposed three kinds of derivations or approximations that may be used. Those which are "component-local" and determine a value using only the history data for that particular attribute; those which are "object-local" and determine a value using the values or derivation functions from other attributes of the entity; and those which are "global" and make use of data from the attributes of other entities as well.

4.5 SUMMARY

A data language to manipulate and access time-varying data is an essential component of the temporal data model. The query language extensions should support both "as of" and "walk through time" queries on the data. Temporal data models which support the valid time dimension must also provide the capability to perform retroactive and proactive updates. Some proposals have enhanced the basic relational algebra operations of SELECT, PROJECT and JOIN while others have added extensions to a higher level query language such as SQL and QUEL.

Proposals to enhance the basic relational algebra operations of SELECT, PROJECT and JOIN involved extending them to work over the time dimension in addition to the attribute and tuple dimensions. For example, the SELECT has been split up into a SELECT-IF and SELECT-WHEN [11], to perform a selection along the attribute, and attribute and time dimensions, respectively. A TIME-JOIN [11] has also been proposed to join tuples over the time and attribute dimensions. New basic operations have also been added which function over the time domain. The TIME-SLICE operation has been added to reduce a relation

along the time dimension and the WHEN operation has been provided to return a set of times when conditions are met [11]. Tansel [46] has also added several additional relational algebra operations, such as PACK, T-DEC and DROP-TIME, which are used specifically to convert the non-first normal form relations in his temporal data model to normalized operations to be manipulated by the standard SELECT, PROJECT and JOIN operations.

The SQL and QUEL language extensions have involved the addition of new clauses and functions to the existing data retrieval statements (ie. SELECT and RETRIEVE) rather than the addition of new statements. Additional clauses, such as the WHEN [42], AS OF [7,42], SUBJECT TO [8], FROMTIME [35,36,37], have been added to manipulate the time dimension of the data separate from the manipulation of the data attributes. The addition of functions and operators to be used within existing clauses have also been proposed. These include the VERSION keyword [2], MEET and OVERLAP functions [35,36,37] and the logical and temporal versions of the comparison and boolean operations [8]. The approaches which use separate clauses for temporal manipulation are more easily understood than those which embed temporal operations in the existing clauses.

The temporal data language proposed for the temporal specific data model based, on time sequence collections [39,40], was not based on the standard relational algebra operations. These operations were unique in that they manipulated entire time sequence collections with operations such as AGGREGATE, ACCUMULATE, RESTRICT and COMPOSE.

Also introduced in this chapter was the requirement for derivation and approximation functions for queries on temporal data which is not explicitly recorded in the database. The derivation or approximation of data values is required for continuous temporal data when specific point in time queries are required. Though not specifically part of the temporal language, these functions are required at an implementation level to support the functions of the language.

At this point, all of the components of a temporal data model, the data structures and the data language, have been presented. Following the development of a data model, a database management system to support this model must be developed. Chapter 5 will introduce the issues to be addressed in the development of a temporal database management system and present some of the proposals for the implementation of the temporal data structures.

5. PHYSICAL DATA MODEL

5.1 INTRODUCTION

Research on temporal databases to date has focused on the development of the temporal data model and language extensions to support manipulation of temporal data rather than the physical implementation. Even though research on temporal databases has been ongoing since the early Seventies (Mckenzie [31]), very few of the models have been implemented due to data access and storage issues associated with temporal data. Recent advances in data access techniques and mass storage technology though, have made the implementation of a temporal database feasible.

Most of the implementations to date have built a temporal database on top of an existing DBMS (Klopprogge and Lockemann 26], Adiba and Quang [2], Snodgrass and Ahn [4], Abbod, Brown and Noble [1]). These prototypes have been important in pinpointing ineffeciencies in existing DBMSs for supporting temporal data. An implementation of the POSTGRES data model (Rowe and Stonebraker [33]) is the only one which has not been built using an existing DBMS.

This chapter will outline the several implementation strategies which attempt to meet the unique design goals associated with temporal data (Section 5.2). Section 5.3 covers the two main issues associated with implementing a temporal data model, storage management and efficient data access. This is followed by a discussion of some of the proposed implementations of temporal databases in Section 5.4.

5.2 DESIGN GOALS

There are several goals unique to the implementation of temporal databases (Dadum, Lum and Werner [14]). These include:

(1) fast access to the current version. Though the database is storing historical data, most queries on the database are expected to be against the current version of the data (Dadam, Lum, Werner[14]) so access should be most efficient to the current version.

(2) on-line access to historic data. Queries against historic data via *ad hoc* queries must be supported by maintaining a reasonable volume of data online for on demand access. This implies that the historic data should be stored as compactly as possible to maximize the storage utilization, but still support efficient retrieval.

(3) selective versioning. History will not be required for all attributes of a tuple and therefore it should not be necessary to store multiple versions of non-time varying data. History should be stored selectively in order to reduce the amount of redundancy in the storage of versions of data. This relates to the previously mentioned fact of maximizing storage utilization.

(4) changes to the relational schema. Relational databases should allow dynamic modification of the database schema in the form of adding or deleting attributes, in such a way that historic data is still valid.

(5) support dynamic data. The assumption is made that the volume of temporal data will continue to grow over time and the DBMS will continue to support efficient retrieval and maximize storage utilization.

5.3 IMPLEMENTATION ISSUES

The two main implementation issues for temporal databases are those of storage management and efficient data access. (Ahn and Snodgrass [5])

5.3.1 Storage Requirements

The most obvious issue is the continuously increasing storage requirements. The size of a temporal database will never decrease because update operations will be translated into inserts, and delete operations will perform only logical deletions by flagging a tuple as deleted. Even with the use of compression techniques on the data, the amount of data to be maintained will still be very large.

Ahn [3] has compared tuple and attribute versioning in terms of storage requirements. Tuple versioning has the disadvantage of added redundancy resulting from the duplication on non-time varying attributes. Attribute versioning solves this problem but has the additional overhead and complexity of linking each attribute version to the base tuple.

Ahn has compared the storage requirements of tuple and attribute versioning to determine the number of updates required per tuple before attributing version is favored over tuple versioning. In general, it was found that the number of tuple updates required before attribute versioning is favored is directly proportional to the overhead for each attribute version (ie. pointers) and inversely proportional to the average size of the time varying attributes. That is, the larger the overhead for each attribute version, the

more updates required to the tuple before attribute versioning is favored over tuple versioning. As well, the larger the size of the time varying attributes themselves, the fewer updates required to favor attribute over tuple versioning.

Not only is the volume of data always increasing, but it is important to maintain a large volume of data online to support *ad hoc* queries. Recently, optical disks [10,22,25,45] have been introduced to store large quantities of data online and may provide a solution to the storage of historic data. Current secondary storage technology, the magnetic disk, provides online direct access capability, but have a storage capacity which is too small for the volume of data in a large temporal database.

Developments in optical storage devices have presented this technology as a viable solution to the storage problem associated with temporal databases. The advantages of optical disks include:

- low cost per bit of stored information
- long archival life
- high storage density
- direct access capability

Although optical disks are approaching magnetic disk in terms of capabilities, there are important differences. These include the fact that optical disks currently provide write-once, read many (WORM) capability and the access times are much slower than for magnetic disk. The write-once capability of the optical disks is not be a limitation for its application in temporal databases since they can be used to archive data which will no longer be updated. The write-once capability is a problem if a dynamic access method is used since these access methods usually require reorganizing or rewriting data to maintain efficient access.

5.3.2 Access Requirements

In addition to the problems of supporting a large volume of data online, there is the associated problem of efficient access to this data. Efficient access methods are required to access a continuously expanding volume of data with a large number of duplicate primary key values. Recall that the primary key in a temporal database uniquely identifies a tuple within a snapshot, not within the entire temporal database. Therefore, the primary key does not contain a timestamp value and duplicate primary key values result because updates generate new versions of tuples with the same primary key.

Access methods can be divided into two groups, static and dynamic. Static access methods are designed for files with a relatively stable size. Dynamic access methods are designed to support file growth without loss of performance and include such techniques as dynamic hashing and grid files. Most temporal databases are dynamic and thus require a dynamic access method which will provide efficient access to the current version of the data. With static temporal data, the distribution of the data is already known, and it may be possible to optimize the storage structure of the data before queries are run against it.

The non-uniqueness of the keys in temporal databases also causes problems in providing efficient data access. The non-uniqueness results because updates generate new versions of tuples with the same primary key, assuming the timestamp associated with the data is not made part of the key. Static access methods do not work well even with static temporal databases due to long chains of tuples with the same key. Even dynamic access methods are not designed to support long chains of tuples with the same key. Long chains of tuples with non-unique keys results in performance degradation for access to all data, both current and historic.

Enforcing uniqueness by adding the timestamp to the key is not a feasible solution to the non-uniqueness problem since this introduces the problems associated with handling a multikey field. One problem is that

exact match queries may no longer be possible because the timestamps may not be known, or the granularity of the timestamp may be too low (ie. seconds) to make exact matches possible. If the timestamp is made part of the primary key, the access method chosen must support range queries. This elminates most hashing algorithms from consideration.

5.4 IMPLEMENTATION APPROACHES

In this section, several proposals for the implementation of temporal databases will be discussed. These include implementations using the existing relational model, delta versioning, two-level storage structures, multidimensional file partitioning, Time-split B-trees and the POSTGRES implementation.

5.4.1 Implementation Using the Relational Data Model

Some of the temporal data models proposed in Chapter 3 have been implemented "on top" of an existing relational database management system. This has been done by adding timestamp attributes to each relation and a layer of application code between the application program and the DBMS to translate temporal queries into standard queries on the underlying DBMS. Knowing that this is not the optimal implementation due to the performance issues described in the previous section, the "on top" implementation is intended primarily as a prototype system to identify problems with conventional access methods and query processing algorithms when supporting temporal data (Snodgrass and Ahn [4,43]). The problems identified in these implementations include the issues associated with storage management and efficient data access desribed in Section 5.3. Another intent of the "on top" approach was to develop a temporal front-end that could easily be ported to other DBMSs (Abbod, Brown and Noble [1]).

5.4.2 "Delta" Versioning

Storing a complete copy for each version of a tuple or attribute at each new database state would generate a very large volume of data over time. If tuple versioning is used, a lot of non-time varying attribute redundancy would result. One method proposed for storing successive versions of temporal data is to use "delta versioning" (Dadum, Lum and Werner [14]) or "differential files" (Ahn and Snodgrass [5]). In this method, one complete version is stored and the remaining ones stored as differences or "deltas" from the complete version.

Dadam, Lum and Werner [14] have outlined four variations of delta versioning. The first two are forms of "forward oriented versioning" in which more recent states are derived from older states. The last two use "backward oriented versioning" in which the current state is used to derive older versions.

In the first variation, the initial version of the data is the complete version and each successive delta stores the differences from the version immediately preceeding it. This method provides slow access to the current, or most recent version, since it must be derived by applying all of the deltas to the initial version. This will not provide fast access to the current version, which is one of the design goals of temporal DBMSs.

The second variation also stores the initial version as the complete version. In this case though, each subsequent version is stored as a delta from the initial version, rather than from the previous version. This method provides faster access to the current version since it can now be generated by applying a single delta record to the initial complete version.

One problem associated with both of these forward oriented versioning schemes is the purging of old temporal data. If older temporal occurrences are removed, the complete initial version would be removed.

This would require generating a new complete initial version and modifying some delta versions. Alternatively, a new complete version could be generated at regular intervals by combining all deltas, and then basing all subsequent versions on deltas from this new complete version.

The next two variations are based on backward oriented versioning. In both of these methods, the current version of the data is stored as the complete version. In method three, each version is stored as a delta of the next newer version. That is, when a new version is added, the previous version becomes a delta containing the differences from the new current verison. In method four, each previous version is a delta of the current version. This is the worst alternative since each delta must be regenerated whenever a new version is added.

Dadum, Lum and Werner have suggested that method three is the best alternative. In this variation, access is fastest to the current version and slower for each older version. As well, purging of old temporal occurrences does not require regenerating any deltas or initial complete versions. This method is also referred to as a "reverse differential file" (Ahn and Snodgrass [5]). This method could be used effectively with a combination of optical and magnetic storage devices if the data was organized so that the current version is on magnetic disk and the deltas are on optical disk. Delta versioning in this manner fits well into Snodgrass and Ahn's "temporally partitioned storage structure" outlined in the following section. On the other hand, if the organization places the deltas on an optical device, the purging of old temporal occurrences would not be allowed since optical disks are currently write-once devices.

5.4.3 Two Level Storage Structures

Several researchers (Ahn and Snodgrass [5], Ahn [3],Adiba and Quang [2], Sarda [15]) have suggested the use of a two level or "temporally partitioned" [5] storage structure for dynamic temporal data. The

implementation of Time-Split B-trees (Lomet and Salzberg [27,28]) and POSTGRES (Rowe and Stonebraker [33]) also propose the use of a two level storage structure. These approaches will be discussed in subsequent sections since they propose specific use of this storage structure. In a two level storage structure, the current data is separated from the historic data, based on the different characteristics of the two types of data. Snodgrass and Ahn have done the most research into the implementation of this scheme and their findings form the basis of the following discussion.

As mentioned above, the characteristics of current and historic data are distinct. Current data refers to the most recent temporal occurrence while historic data refers the all of the remaining temporal occurrences. The current data differs from historic data in that a) there is always only one temporal occurrence which is current data, b) current data is accessed more frequently, c) current data can be updated, and d) the number of current tuples tends to stabilize over time. Historic data, on the other hand, has a) a continually growing number of occurrences for each primary key, b) data that is not updated, only added to, and c) data that is accessed less frequently than current data.

For these reasons, Ahn and Snodgrass [5] have defined a "temporally partitioned store" in which the data is divided into a "current store" and a "history store". The current store contains all of the current data and possibly some recent historic data. The history store contains the rest of the historic data. All non-temporal queries which access only current data can be satisfied entirely by the data in current store.

Since the current and history stores are separate, they can be implemented using different access methods, storage structures, and storage devices, optimized for their unique characteristics. The important components of a two level storage structure are the implementation of the current and history stores and the split criteria for partitioning the data between the two stores.

5.4.3.1 Current Store

The current store is intended to hold the current version of the tuples and possibly one or two previous versions. The volume of data will be smaller than in the history store and the number of tuples will tend to stabilize over time. This being the case, it is much like a conventional database and conventional access methods and storage structures can be used and still provide acceptable performance. By separating the current and historic data in this way, access times for current data will not be affected by a large volume of historic data. The current store will also usually be stored on a magnetic storage device.

5.4.3.2 History Store

Since it is physically separate from the current store, the implementation of the history store can be optimized for the characteristics of the historic data . The characteristics of historic data are that there may be multiple temporal occurrences for each primary key, it can not be updated, it is continually growing since it is only added to, and it is much less frequently accessed than current data. Online access is still required though, so data access times must be acceptable. Depending on the storage structure of the historic data, it may be possible to use to an optical storage device to take advantage of the benefits of that medium.

Several methods for the structure of the history store have been proposed by Ahn and Snodgrass [5] and Ahn [3]. These include (a) reverse chaining, (b) accession lists, (c) clustering, (d) stacked versions and (e) cellular chaining. (Figure 5.1)

(a) Reverse Chaining

In reverse chaining (Figure 5.1a), the tuples are chained in reverse time sequence in the history store. That

is, each tuple has a pointer which points to its predecessor. The tuple for a given key in the current store points to the first tuple in the chain of tuples in the history store. Each tuple chain is added to only at the beginning and no updating of existing tuples in the history store is required. This method makes the history store suitable for write-once storage devices. Retrieval of historic data is performed by following the tuple chain until the version with the required timestamp is located. This method can be used for either tuple or attribute versioning.

(b) Accession Lists

The use of accession lists was proposed to eliminate traversing the long tuple chains in the history which result when reverse chaining is used. In this method, an accession list is established between the current and history stores which point to the historic versions of a tuple in the history store. (Figure 5.1b) The tuple in the current store has a pointer which points to the accession list. The accession list is essentially an index into the history store which contains a pointer to each historic version of the tuple, and

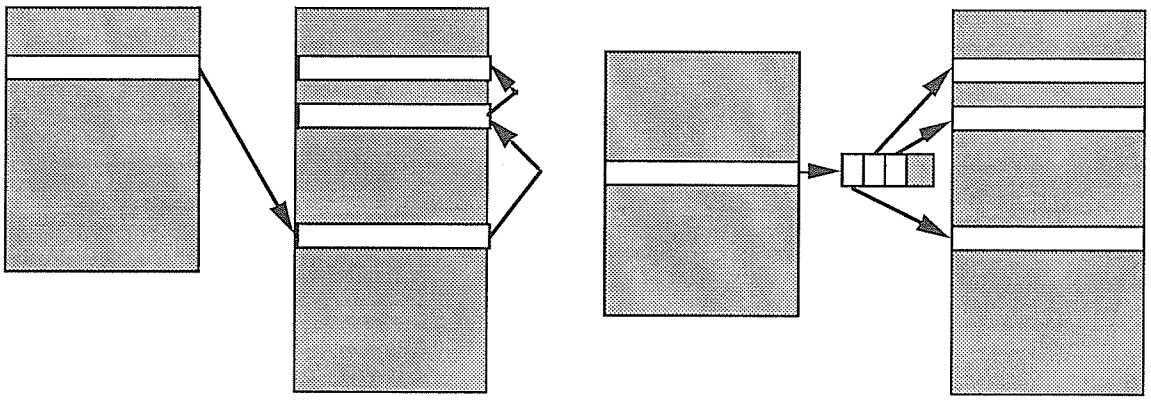


FIGURE 5.1 TWO LEVEL STORAGE STRUCTURES (Snodgrass and Ahn)

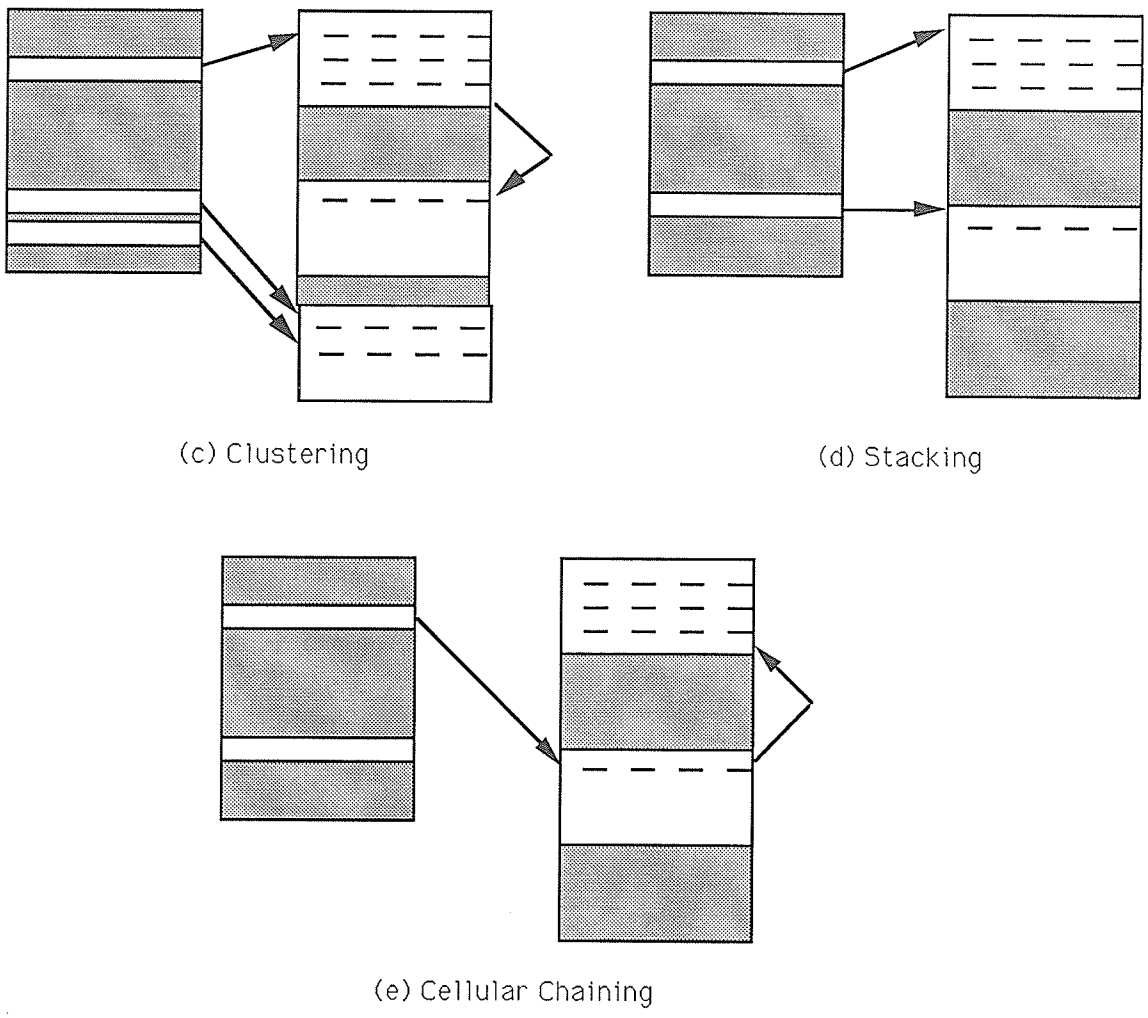


FIGURE 5.1 TWO LEVEL STORAGE STRUCTURES (Snodgrass and Ahn)

the temporal information associated with the version. Storing temporal information in the accession list may reduce the number of reads to the history store since tuples can be eliminated from the search by simply examining this temporal information. This also allows some temporal queries, such as inquiring only about the existence of a tuple at a point in time, to be satisfied without actually retrieving the tuple. Accession lists could be used with attribute or tuple versioning. In attribute versioning, an accession list

would be required for each temporal attribute in each tuple; this may result in excessive overhead, making this scheme unusable.

(c) Clustering

The problem with reverse chaining and accession lists is that with successive inserts, the historic versions for a particluar tuple can become spread over many pages in the history store. This will result in many I/Os when querying all of the historic data for the tuple with this primary key. With clustering, the versions related to a primary key are stored in the fewest blocks possible to minimize disk accesses (Figure 5.1c). When an entire page is allocated for each primary key though, poor storage utilitization can result. Therefore, to maximize storage utilization, the historic versions of several tuples share a single page. When a page overflows, a page split is performed and tuples are moved and pointers updated. Thus, disk I/O is minimized and storage utilization maximized. The possibility of using a write-once storage device has been eliminated though, due to the page split processing.

(d) Stacking

The stacking method differs from the previous methods in that each tuple has a maximum number of historic versions in the history store (Figure 5.1d). When a new version is added to history and the maximum number of versions has been reached, the newest version is written over the oldest version. The oldest version has then been lost. Since the number of versions to be kept is known, and is the same for all tuples, space can be preallocated when the first version is inserted into the history store and all versions can then be clustered together. This could result in poor storage utilization though, if space is preallocated for a maximum number of versions which is more than required for most tuples. In addition, the fact that old versions are overwritten eliminates the possibility of using write-once storage devices.

(e) Cellular Chaining

Cellular chaining is a combination of several of the previous methods (figure 5.1e). When the first version of a tuple is inserted into the history store, a "cell" is allocated. A "cell" is a preallocated area which can hold a fixed number of versions of a tuple, as in the stacking approach. Unlike the stacking method, when a cell becomes full, a new cell is allocated and the next new historic version is inserted into the new cell. The new cell is chained to the previous one, as the tuples were in reverse chaining. Cellular chaining can be used for both tuple and attribute versioning, and since insertions are always appended to the end of the historic store, this organization can be used on write-once storage devices.

5.4.3.3 Split Criteria

The split criteria governs the movement of data between the current and history stores. In general, the current store will contain only the current version of the data. With a temporal database, the current version may be based on, the transaction time, the valid time or both. And with temporal databases, retroactive and proactive changes may be made to further complicate matters.

Ahn and Snodgrass [5] have proposed that the current store only hold tuples with currently undefined "valid to" and "transaction stop" times. That is, only those tuples which have not been replaced by a new version nor deleted. In addition, a third store, the "archival store", has been suggested which only contains tuples with a "transaction stop" time (ie. tuples that have been deleted). This store would only be required to support queries on the state of database as of some moment in the past (ie. rollback queries) and because no further updates will be done to these tuples, they could be stored on a WORM device. It is also possible to store some frequently accessed history data in the current store to minimize I/O to the history store, or to store less frequently accessed current versions of the data in the history store.

Thus, the split criteria can depend on several factors including the number of stores, the volume of current data, and the access patterns to the current and history data.

5.4.3.4 Indexing

Indexes are required in temporal databases to improve the efficiency of data access, as they are in non-temporal databases.

The primary index of a non-temporal database consists of entries containing a key value and a pointer. In a temporal database, each index entry is extended to include a list of pointers which point to the current version and to each historic version stored in the history store. In addition, some temporal information, such as valid and transaction times could be stored in each index entry to reduce the number of accesses to the history store and so that some temporal queries can be processed without accessing the actual data. The advantage of this index organization is that all historic versions are indexed and the temporal information stored in the index can be used to improve the performance of some queries. The size of the index though, will continue to grow and may slow access to the index entries for the current data.

Another option is to maintain only the pointer to the current version and one historic version pointer in each index entry. The historic pointer could point to the first entry in the chain of history tuples or attributes. With this index structure, current data index entries will not be affected by a growing number of historic data index entries, but not all historic versions are indexed.

A third option for primary indexes is to partition the index similarly to the data to create a "temporally partitioned index". In this method the index entries for the current store are separated from the historic store index entries. The current index store could maintain a pointer to the current version as well as a historic pointer, which points to an accession list for historic versions. The advantage of this method is that

the performance of the more frequent current data queries will not be affected by the number of historic version entries and all historic versions will still be indexed.

Often, secondary indexes in a non-temporal database are non-unique indexes, and hence differ from primary indexes in that each entry is not expected to be unique and point to only a single tuple. In a temporal relation, secondary indexes will generally be much larger than in non-temporal relations due to the number of historic versions. In temporal databases, the secondary index itself can be a temporal relation in which each entry is supplemented with the valid and transaction time information. This allows temporal predicates to be satisified before the data is accessed to minimize false hits. It is also possible for the secondary index to contain only partial temporal information and be stored as snapshot, rollback (supporting transaction times only) or historic (supporting valid times only) relations.

5.4.4 Multidimensional File Partitioning

A multidimensional file partitioning scheme has been proposed by Rotem and Segev [34] for the implementation of the temporal data model based on time sequences and time sequence collections (Shoshani and Kawagoe [40], Segev and Shoshani [39]).

A multidimensional file partitioning (MDFP) scheme is one which is used when access to a file is required by more than one attribute, and sequential access and range queries using these attributes are frequent. This means that the data should not only be indexed by these attributes but should physically be stored in this sequence in order to minimize I/O. In a temporal database, the tuple's key value and time attributes are the attributes on which the file should be indexed.

A MDFP organization has three levels. At the highest level is information regarding the partition points of the data. In the MDFP scheme, a range a values for the indexed attribute is grouped into a "segment" and the intersection of the segments for two attributes forms a "cell". If the employee number is an indexed

attribute, the employee number range 0 to 100 may be in the first segment of employee numbers. If the valid time is the second indexed attribute, the time period from January 1 to January 31 may be the first segment of the valid times. A cell is then formed containing all temporal occurrences for employees 0 to 100 from January 1 to January 31. At the next level in this organization is a directory with one entry per cell containing a pointer to the data page on the storage device where this cell is stored. At the lowest level is the data page itself containing the tuples.

Recall that the temporal model based on time sequences was represented "conceptually" as a matrix where each row represented a time sequence for a primary key value. In this case, it is best to partition the file by the key value and the time attribute since queries will often deal with the value of the key or a group of keys over a period of time. It is also assumed that direct access to a time sequence by primary key is required, and in some cases sequential access by this value is needed. The implementation problem in this case is how to partition the data into cells to support these access patterns and minimize cell overflow.

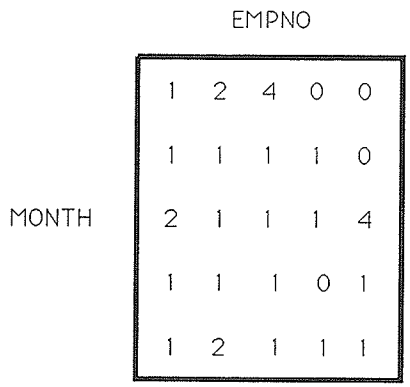


FIGURE 5.2a FREQUENCY MATRIX

Two methods have been proposed to do the partitioning, "symmetric" and "asymmetric partitioning" (Rotem and Segev [34]). In both of these methods, a "frequency matrix" which stores the number of temporal occurrences related to a primary key during a given time period, is used as input to the

partitioning algorithm. The frequency matrix in Figure 5.2a indicates the number of occurrences of a temporal attribute for each employee, in each month. The first row represents the first month, the second row the second, etc. The partitioning schemes subdivide the frequency matrix into cells, and tuples corresponding to the temporal occurrences in each cell are stored in a single physical page.

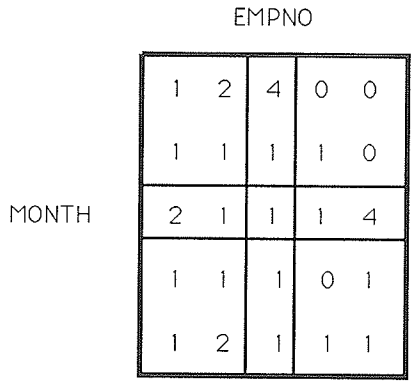


FIGURE 5.2b SYMMETRIC PARTITIONING OF FREQUENCY MATRIX

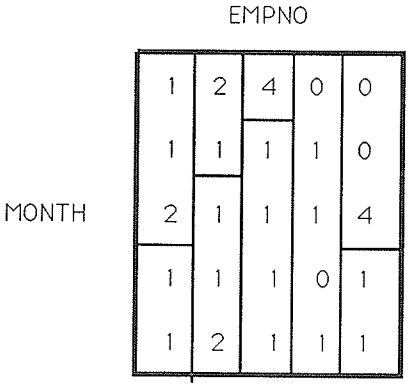


FIGURE 5.2c ASYMMETRIC PARTITIONING OF FREQUENCY MATRIX

In symmetric partitioning, the time sequence array is partitioned symmetrically using vertical and horizontal grid lines. (Figure 5.2b) In asymmetric partitioning, the data is partitioned by grid lines vertically into segments, and each segment is then partitioned horizontally independent of the other

segments in order to reduce the number of page overflows. (Figure 5.2c) The advantage of symmetric, over asymmetric partitioning, is that the amount of data stored about the partitioning itself is much less since the horizontal partitions are the same for the entire grid. Asymmetric partitioning is better in terms of reducing overflows; the reduction in overflow data can easily outweigh the overhead of additional storage requirements for maintaining information regarding the partitioning. In the asymmetric case, the data is partitioned into segments over the primary key values rather than the time values since it is assumed that most queries will access the key values over a time range. This method will cluster time values together for a primary key, which will reduce I/O for this type of query.

Shoshani and Kawagoe [40] have taken different approaches to implementing data structures for time sequences that support regular data and those that support irregular data. The implementation of regular time sequence arrays for static data is the simplest case. In this case, a value exists for each key value at each time point. The time interval is regular, allowing the correct "column" to be calculated given a time value. Random access over the keys can be provided by an index which determines the ordinal ("row") position. The array can be partitioned into cells and a tuple located by using "array linearization" to first determine the correct cell, then to locate the position within the cell. The dynamic case is an extension of this in which the current cells are added to, for all keys, at the same regular interval and previous cells are treated as static.

The implementation of an irregular time sequence array is not as simple due its sparse array. This could be implemented as described above but the large number of null values would waste storage and result in slow access times. In the static case, a technique called "header compression" could be used. In this method, a sequence of the counts of the null and non-null values is kept in a separate header which can then be accessed using array linearization to locate non-null values. If the time sequence array is modelling event data where only a boolean value is required to indicate if an event did or did not occur, then only the header itself is required.

In the case of irregular time sequences modelling dynamic data, each key is assigned a page to which its versions are added to at irregular time intervals. The number of pages may differ for each key value, and they may have different start times, end times, and durations. To handle this, an index could be created which stores the start time and page number for each key, but accessing data over the time domain then becomes difficult. This problem can be solved by creating a grid file [29] to access the pages by start time and duration.

5.4.5 Time-Split B-Trees

A Time-Split B-tree (Lomet and Salzberg [27,28]) is an access method for temporal data based on Write-Once B-trees (Easton [20]). This approach has been designed to support rollback databases in which the data is timestamped with the transaction time. A Time-Split B-tree is a two-level storage structure in which data is migrated automatically from the current data store to the historic data store. The current data is stored on magnetic disk and the historic data can be stored on any type of random access device, including write-once storage devices.

As in the B^+-tree, the data records in the Time-Split B-tree are stored in the leaf nodes. The index nodes though, contain a timestamp in addition to the low key value from the lower level node. When traversing the tree, it is necessary to find not only the correct key value, but also the correct timestamp. The traversal is done by searching index entries for the largest key and timestamp pair in the index node which do not exceed the search key and timestamp, and choosing the last one in the node that satisfies this criteria. The pointer corresponding to this node is followed and the search process repeated until the data node is reached. For example, when searching the second version of the tree in Figure 5.3a for a key value of 4000 and T=10, the pointer from the second index entry (3000 T=1) is chosen since both the key and timestamp value are less than the search values, and this is the last entry in the node for which this criteria is satisfied. Choosing the last entry in the node for which the criteria is satisfied implies that this is path will have the

most recent version of the data for this key value and timestamp.

The Time-Split B-tree is unique in its splitting algorithm. In this algorithm, only nodes in the current data store, which is on magnetic disk, can be split. It is during some of these splits that nodes migrate to the historic data store. When splitting an overflowing data node, there are two possible alternatives, a "key split" or a "time split".

When a key split is done, the records with keys lower than the split value are kept in the old node and the others are moved to a new node (Figure 5.3a).

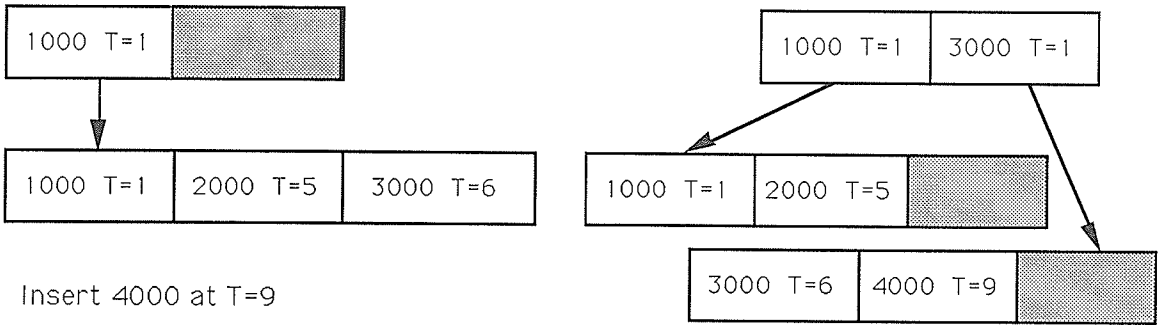


FIGURE 5.3a TIME-SPLIT B-TREE - KEY SPLIT

When a time split is done, a split time value is chosen and all those records with timestamps less than the split time remain in the old node and those with greater or equal timestamps are placed in a new node. Those records which have a timestamp less than the split time, but are still valid at the split time, have a copy in both the old and new nodes. Thus, some redundancy may result when a time split is done. At this point though, the old node can migrate to the historical data store since no further updates will be made to this node (Figure 5.3b).

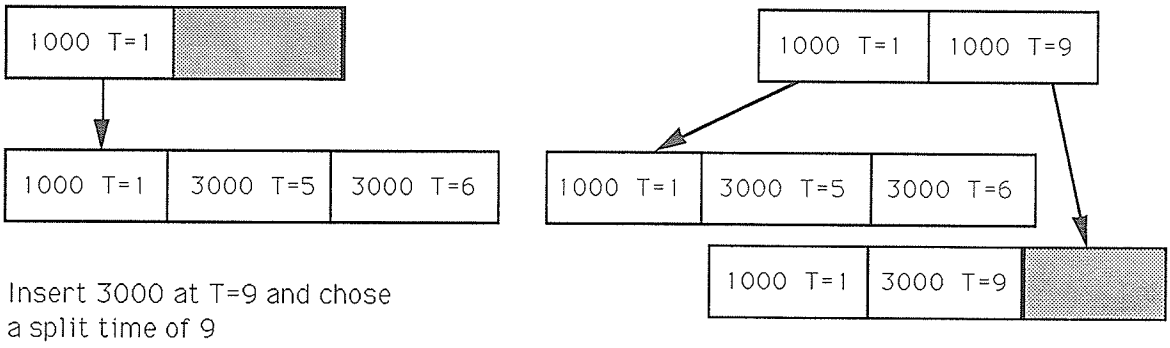


FIGURE 5.3b TIME-SPLIT B-TREE - TIME SPLIT

When data nodes are split, index nodes must also be updated. When a key split is done, the new index entry has the low key value of the new node and a timestamp which is the same as the one in the index entry pointing to the old node. When a time split is done, the new index entry has the low key value of the old node, and the timestamp is the split time. It may also be necessary to split index entries, which is done in a similar manner as data node splits.

The DBMS must make a choice at split time whether to do a key split or a time split. If a time split is chosen, a split time must also be determined. The choice between a key and time split depends on the ratio of current to historic data in the node. In general, the more historical data present, the better a time split would be. The choice of the split time will affect the amount of space used in the current (new) node, the amount of space used in the historic (old) node and the amount of redundancy. In Figure 5.3b, a split time of T=7 would have generated greater redundancy since the tuple with "3000 T=6 " would also have been placed in the new node.

When using Time-Split B-trees, the size of the historic nodes can vary as long as the length of the node is stored with the node address in the index. The history store can then be implemented as a sequential file, and when a node migrates to history, it can simply be appended to the end. This will eliminate poor storage

utilization which may result when using a fixed blocked write-once storage device.

Secondary indexes can also be implemented as Time Split B-Trees. In this case, each record contains a timestamp, a secondary index value and the primary key. The secondary index is maintained independently of the primary index and must be updated whenever a secondary index field is updated or a new record is added. The data records associated with the secondary key value can be retrieved using the primary key and timestamp values found as a result of searching the secondary index tree. With such a structure, it may also be possible to handle queries involving only the secondary index fields without going to the data itself.

The proposed Time-Split B-trees have been simulated in order to determine the effect of splitting policies on storage utilization [28]. In this simulation, three splitting policies were compared. The first was the Write-Once B-Tree (WOBT) policy where a time split is always done using a split time of the current time. The second split policy was the Time of Last Update policy where a time split is always done if there is historic data, and the split time chosen is the time of the last update in the node. This split time is chosen to optimize situations in which inserts are done after an update to a record in a node. The split time does this by reducing the size of the historic nodes by minimizing the amount of current data in this historic node. Chosing this split time causes the data inserted after the last update to be moved to the new node and not stored in the historic node. This policy is better than the WOBT policy if the cost of the WORM storage for the historic data is greater than a factor of ten cheaper than magnetic data since less historic data will generated.

The third split policy compared is the Isolated Key Split policy, in which a key split is done if two thirds or more of the data in a node is current data. Otherwise a time split is done with the time of last update as the split time. Key splits are done with the other split policies only if there is absolutely no historic data in the node. Performing a key split though, with very little historic data in a node, will not generate a new historic node and will often create less redundancy in the current database. This policy is the best choice if the cost of WORM storage is less than a factor of ten cheaper than magnetic storage since there is a substantial

reduction in the size of the historical database as compared to using the other two split policies.

5.4.6 POSTGRES Implementation

The POSTGRES data base management system is a research system which includes support for "historical" data (Stonebraker [44]). The POSTGRES database does not support historical databases, but rollback databases in which data is recorded by transaction time.

The POSTGRES implementation of "historical" versions is done with forward oriented delta versioning of tuples. In this implementation, the initial tuple is the complete version, called the "anchor point", and "delta records", containing only the changed data, are chained from the anchor point . This method was chosen over a backwards oriented versioning scheme based on the assumption that most delta records will be relatively small and will be stored on the same physical page as the anchor point. Data access is done by starting at the anchor point and applying the delta versions to generate the desired version.

POSTGRES uses a two level storage structure to separate current and historic data. In order to prevent the number of delta records for a tuple from growing too large, an asynchronous task call the "vacuum cleaner" is run which migrates records from the current data store to a history store called the archive storage area. This task will move the anchor point and all the delta records to the archive storage area and generate a new anchor point, which will remain in the current data store. The time interval between the executions of the vacuum cleaner process is defined with a relation. This process produces an organization similar to the two level storage structure proposed by Snodgrass and Ahn, except that in POSTGRES, the migration of data from the current data store to the archive is only done periodically, rather than at the creation of each historical version. The disadvantage of this method of migrating data is that there is historic data in the current data store and if the period is long between migrations, access to the current data may be adversely affected by a large number of delta versions.

Since the archival storage system is separate from the current data store, the two storage structures may be different. The archive though, must still be available online for random access. In the archive, the anchor point and all its delta records are written as a single variable length record. These records are never updated, so a write-once storage device, such as optical disks, may be used. The access patterns can be different for historic data and therefore separate indexes may be required for the archive. Thus, for queries which access historic data, the query optimizer may chose two access paths, one for the current data store and one for the archive.

5.5 SUMMARY

A number of proposals have been reviewed for the physical implementation of temporal databases. These proposals have attempted to meet the goals of providing efficient access to the current version of the data, supporting online access to historic data, and allowing selective versioning of data. The two main issues facing the development of a temporal DBMS are those of storage requirements and efficient data access, both of which are related to the fact that the volume of data in a temporal database is constantly increasing.

The majority of implementation proposals have handled this by partitioning current and historic data into a two level or temporally partitioned storage structure (Ahn and Snodgrass [5], Adiba and Quang [2], Sarda [35], Stonebraker [44], Lomet and Salzberg [27]). Snodgrass and Ahn have proposed several organizations for a two level storage organization, in particular for the structure of the history store for historic data. The Time-Split B-Tree (Lomet and Salzberg [27]), is also based on a two level storage structure in which data migrates automatically, during some node splits, from the current to the historic data stores.

Several proposals have been made for using a two level structure, but only the POSTGRES DBMS has actually been implemented. POSTGRES uses a form of forward oriented delta versioning in order to selectively version data and maximize storage utilization. In this implementation, an asynchronous process

is periodically run to combine deltas to create a new initial version and migrate older deltas to the history store. Though POSTGRES has used forward oriented versioning, Dadum, Lum and Werner [14] have found that a form a backward oriented version in which each version stored as a delta of the next newer version would be the best form of delta versioning. Using this method, access is fastest to the current version and purging of old temporal occurrences does not require regenerating any new deltas or initial versions.

All implementations which use a two level storage structure suggest the use of a magnetic storage device for the current data store and an optical storage device for historic data. This has been done to take advantage of the benefits of on-line optical storage devices for large volume, lower cost-per-byte storage for data which will not be updated once it has been written.

Rotem and Segev [34] have proposed a storage structure for the temporal specific data model [39,40] based on time sequences and time sequence arrays. Their storage structure uses a multidimensional file partitioning scheme to optimize access to the data by primary key and time attibutes. In their scheme, a frequency matrix which keeps track of the number of temporal occurrences, is used as input to an algorithm to partition the time sequence array to provide efficient access and minimize partition overflow.

None the implementation proposals, except for POSTGRES, have been implemented as part of a DBMS. Several of the schemes, such as two-level storage structures, time-split B-trees and multidimensional file partitioning, have been simulated to predict their handling of temporal data. As the need for temporal data support grows and storage technologies such as optical disks continue to improve, more implementations of temporal DBMSs will emerge, likely based on the two-level storage structures proposed by Snodgrass and Ahn and Lomet and Salzberg's Time-Split B-Trees.

6. SUMMARY

The intent of this thesis was to present a comparison of data models supporting temporal data. The temporal data model consists of the data structures to support temporal data and the data language to manipulate the data structures. A discussion of the physical implementation of the data structures of the temporal data model was also presented.

The development of a temporal data model, and the implementation of this model in a temporal database management system, has emerged as an important area in database research. This is due to the increasing demands for historical data in reporting and analysis. Applications such as project management, financial records, and decision support systems would benefit from a data base management system which supports and maintains temporal data in a uniform and consistent manner, independent of the application program. The importance of this topic is demonstrated by the fact that recent texts by Date [16] and Elmasri [21] have mentioned temporal data management as an active area of database research.

The temporal data model supports at least one of the two dimensions of time. These two dimensions are the valid time and the transaction time. The valid time is the time the data is valid in the real world. The transaction time is the time the data was recorded in the database. The valid time is of greater interest to a user of the database since it is the real world time associated with the data. This is the time dimension that the user can manipulate. A database administrator though, requires a transaction time in order to perform database recovery using backup copies and a transaction log. Four types of databases have been emerged based on these two time dimensions. They are the snapshot database (no time support), the rollback database (supports transaction time), the historical database (supports valid time) and the temporal database (supports both valid and transaction time). Though the temporal database provides the most complete temporal information about the data, a temporal data model supporting a single time dimension should

support the majority of applications requiring temporal data. A temporal DBMS supporting a historical database is most likely to be implemented since the valid time dimension of the data is maintained. The historical database also allows retroactive and proactive updates to data.

Three significant approaches to developing a temporal data model were surveyed in the thesis. Two of the approaches proposed modifications to the relational data model, to support tuple timestamping and attribute timestamping, and the third proposed a temporal specific data model, not based on any existing data model. The temporal specific data model, based on time sequences and time sequence arrays, provided the best support for temporal data since it was developed specifically to support time varying data. Implementing this data model as a DBMS may not be most beneficial though, since the support for non-temporal data is not optimal. Implementations of the temporal data models based on the relational data model are more probable since a more flexible DBMS, providing support for temporal and non-temporal data, would result. Of the two temporal data models based on the relational model, those based on attribute, rather than tuple, timestamping more closely model the characteristics of time varying data. This is the case since it is the attributes values which are varying over time, not the entire tuple. In addition, a temporal data model based on attribute timestamping allows for selective versioning of attributes within a relation. The disadvantage of this type of temporal data model is that attribute timestamping creates relations that are not in first normal form. This results in a more complicated algebra and query language. The use of nested relations, relations in which the attributes can themselves be other relations, have also been proposed as an alternative that provides the advantages of attribute versioning. Therefore, it seems likely that temporal DBMS implementations based on tuple timestamping, using normalized relations, will appear before those supporting attribute timestamping.

The development of a temporal data model and temporal data manipulation languages has been the subject of the majority of the research on temporal data. Few researchers have addressed the physical implementation of these temporal data models, likely since the support of a large volume of historic data online has not been practical. The indicated area of future research on the majority of the temporal data

models is the implementation of the data model. Recent advances in storage technology, such as optical disks, has made online storage of a large volume of data not totally infeasible. This will renew interest on temporal data model implementation and there will likely be prototype implementations, incoporating such techniques as delta versioning and two level storage structures, in the near future. Applications are currently being developed which support time varying data using application maintained timestamps in the database. There is a need for a commercial temporal DBMS. The commercial feasibility of a temporal DBMS will depend on these implementations and on their ability to optimize the use of magnetic and optical disk to provide efficient data access and storage utilization.

Glossary

Attribute. A property of an entity or relationship in an E-R model. Also referred to as a column or field in a relation in the relational data model

Data model. A mechanism to define data and data relationships. Consists of data structures, for defining data objects, and a set of operations to manipulate the data structures.

Database state. A snapshot associated with a transaction time.

Entity. Any identifiable thing for which data must be maintained to support the activities of an organization. The entity may represent an object that tangibly exists, such as a person, place or thing, or a concept, such a project or event.

Historical state. A snapshot associated with a valid time.

Primary key (in the relational data model). Attribute or attributes which uniquely identify a tuple in a relation. Also referred to as a snapshot primary key.

Primary key (in the temporal data model). Attribute or attributes which uniquely identify a tuple in a snapshot from a relation supporting temporal data.

Proactive update. An update to a temporal attribute which will be applied at some point in the future.

Relationship. Defines a connection between entities representing interactions or associations between the objects.

Retroactive update. An update to a historic (ie. non-current) version of a temporal attribute. Also referred to as a correction update.

Entity Relationship model. A representation of real world objects and concepts as entities, relationships and attributes.

Snapshot. The set of all data values that exist for all attributes in a database at a point in time. Also referred to as a database state.

Temporal. Relating to a sequence of time or to a particular time.

Temporal attribute. See temporal data.

Temporal data. A sequence of data values over time. Usually represented as a sequence of data value and timestamp pairs.

Temporal data model. A data model which provides support for temporal data.

Temporal occurrence. A single occurrence of a temporal attribute at a particular point in time.

Transaction time. The time a data value is recorded in a database. Usually the commit time of the insert or update operation.

Valid time. The time a data value recorded in a database is valid in the real world.

Bibliography

1. Abbod, T., K. Brown, and H. Noble. "Providing Time-Related Constraints For Conventional Database Systems." Proceedings of the 13th VLDB Conference. Brighton, England, September 1987. 167-175.

2. Adiba, M., and N. B. Quang. "Historical Mulit-Media Databases." Proceedings of the 12th International VLDB Conference. Kyoto, Japan, August 1986. 63-70.

3. Ahn, Ilsoo. "Towards an Implentation of Database Management Systems with Temporal Support." IEEE 1986 International Conference On Data Engineering. Los Angeles, February 1986. 374-381.

4. Ahn, Ilsoo, and Richard Snodgrass. "Performance Evaluation of a Temporal Database Management System." Proceedings of the ACM SIGMOD International Conference on Management of Data. Washington, D.C., 1986. 96-107.

5. Ahn, Ilsoo, and Richard Snodgrass. "Partitioned Storage for Databases." Information Systems 13 (1988): 369-391.

6. Allen, James F. "Maintaining Knowledge About Temporal Intervals." Communications of the ACM 26 (1983): 832-843.

7. Ariav, Gad. "A Temporally Oriented Data Model." ACM Transactions on Database Systems 11 (1986): 499-527.

8. Bassiouni, M. A. "A Logic For Handling Time in Temporal Databases." Proceedings of the 12th International Computer Software and Applications Conference. 1988. 345-352.

9. Chaudhuri, Surajit. "Temporal Relationships in Databases." Proceedings of the 14th VLDB Conference. Los Angeles, 1988. 160-170.

10. Christodoulakis, Stavros, and Christos Faloutsos. "Design and Performance Considerations for an Optical Disk-Based, Multimedia Object Server." Computer 19.12 (1986): 45-56.

11. Clifford, James and Albert Croker. "The Historical Relational Data Model (HRDM) and Algebra Based on Lifespans." IEEE 1987 Third International Conference on Data Engineering. 1987. 528-537.

12. Clifford, James and David S. Warren. "Formal Semantics for Time in Databases." ACM Transactions on Database Systems 8 (1983): 214-254.

13. Codd, E. F. "A Relational Model For Large Shared Data Banks." Communications of the ACM 13 (1970): 377-387.

14. Dadam, P., V. Lum, and H.-D. Werner. "Integration of Time Versions into a Relational Database System." Proceedings of the 10th International VLDB Conference. Singapore, August 1984. 509-522.

15. Date, C. J. An Introduction to Database Systems. 3rd ed. U.S.A: Addison-Wesley, 1982. 83-95.

16. Date, C. J. An Introduction to Database Systems. 5th ed. U.S.A: Addison-Wesley, 1990. 709-731.

17. Date, C. J. "Dates and Times in IBM SQL: Technical Criticisms." InfoDB 3.1 (Spring 1988): 32-39.

18. Date, C. J. "A Proposal For Adding Date and Time Support to SQL." SIGMOD Record 17.2 (1988): 53-76.

19. Deshpande, V. and P.-A. Larson. "An Algebra for Nested Relations." Research Report CS-87-65. University of Waterloo, December 1987.

20. Easton, Malcom. "Key Sequenced Data Sets on Indelible Storage." IBM Journal of Research and Development 30 (1986): 230-241.

21. Elmasri, Ramez. Fundamentals of Database Systems. Redwood City, California: Benjamin/Cummings, 1989. 648-650.

22. Fujitani, Larry. "Laser Optical Disk: The Coming Revolution in On-Line Storage." Communications of the ACM 27 (1984): 546-554.

23. Gadia, Shashi K. "Toward a Multihomogeneous Model for a Temporal Database." IEEE 1986 International Conference On Data Engineering. Los Angeles, February 1986. 390-397.

24. Gadia, Shashi K. "A Homogenous Relational Model and Query Language for Temporal Databases." ACM Transactions on Database Systems 13 (1988): 418-488.

25. IBM. "RPQ Optical Storage Subsystem Products Announced." Announcement Letter A88-488. June 1988.

26. Klopprogge, Manfred R. and Peter C. Lockemann. "Modelling Information Preserving Databases: Consequences of the Concept of Time." Proceedings of the 9th VLDB Conference. Florence, Italy, 1983. 399-416.

27. Lomet, David and Betty Salzberg. "Access Methods For Multiversion Data." Proceedings of the ACM SIGMOD International Conference on Management of Data. Portland, 1989. 315-324.

28. Lomet, David and Betty Salzberg. "The Performance of a Multiversion Access Method." SIGMOD Record 19 (1990): 353-363.

29. Nievergelt, J., H. Hinterberger and K.C. Sevcik. "The Grid File: An Adaptable, Symmetric Multikey File Structure." ACM Transactions on Database Systems 9 (1984): 38-71.

30. Martin, N. G. , S. B. Navathe and R. Ahmed. "Dealing with Temporal Schema Anomalies in History Databases." Proceedings of the 13th VLDB Conference. Brighton, England, September 1987. 177-184.

31. McKenzie, Edwin. "Bibliography: Temporal Databases." SIGMOD Record 15.4 (1986): 40-52.

32. Ozsoyoglu, Z. M. and L.Y. Yuan. "A New Normal Form for Nested Relations." ACM Transactions on Database Systems 12 (1987): 111-136.

33. Rowe, Lawrence A. and Michael Stonebraker. "The POSTGRES Data Model." Proceedings of the 13th VLDB Conference. Brighton, England, September 1987. 83-96.

34. Rotem, Doron and Arie Segev. "Physical Organization of Temporal Data." IEEE 1987 Third International Conference on Data Engineering. 1987. 547-553.

35. Sarda, N. L. "Modelling of Time and History Data In Database Systems." Proceedings of CIPS Congress 1987. Winnipeg, 1987. 15-20.

36. Sarda, N. L. "Algebra and Query Language for A Historical Data Model." The Computer Journal 33.1 (1990): 11-18.

37. Sarda, Nandlal L. "Extensions to SQL For Historical Databases." IEEE Transactions on Knowledge and Data Engineering 2 (1990): 220-230.

38. Segev, Arie and Himawan Gunadhi. "Event-Join Optimization in Temporal Relational Databases." Proceedings of the 15th International VLDB Conference. Amsterdam, 1989. 205-215.

39. Segev, Arie and Arie Shoshani. "Logical Modelling of Temporal Data." Proceedings of the ACM SIGMOD International Conference on Management of Data. 1986. 454-466.

40. Shoshani, Arie and Kyoji Kawagoe. "Temporal Data Management." Proceedings of the 12th International VLDB Conference. Kyoto, Japan, August 1986. 79-88.

41. Snodgrass, Richard, ed. "Research Concerning Time in Databases: Project Summaries." SIGMOD Record 15.4 (1986): 19-39.

42. Snodgrass, Richard. "The Temporal Query Language TQEL." ACM Transactions on Database Systems 12 (1987): 247-298.

43. Snodgrass, Richard and Ilsoo Ahn. "Temporal Databases." IEEE Computer 19.9 (1986): 35-42.

44. Stonebraker, Michael. "The Design of the POSTGRES Storage System." Proceedings of the 13th VLDB Conference. Brighton, England, September 1987. 289-300.

45. Swords, R. Douglas. "Storage Technologies and The Storage Administrator." Mainframe Journal April 1986: 32-40.

46. Tansel, Abdullah U. "Adding Time Dimension to Relational Model and Extending Relational Algebra." Information Systems 11 (1986): 343-355.

47. Tansel, A. U. and L. Garnett. "Nested Historical Relations." Proceedings of the ACM SIGMOD International Conference on Management of Data. 1989. 284-293.

48. Winsberg, Paul. "A New Approach to Conceptual Database Design." InfoDB 2.4 (Winter 1987/88): 2-7.