AN IMPLEMENTATION OF A TEMPORAL RELATIONAL DATABASE MANAGEMENT SYSTEM

A THESIS SUBMITTED TO
THE DEPARTMENT OF COMPUTER ENGINEERING AND INFORMATION SCIENCE
AND THE INSTITUTE OF ENGINEERING
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

by
Iqbal A. Goralwalla
June 1992

AN IMPLEMENTATION OF A TEMPORAL RELATIONAL DATABASE MANAGEMENT SYSTEM

A THESIS SUBMITTED TO
THE DEPARTMENT OF COMPUTER ENGINEERING AND INFORMATION SCIENCE
AND THE INSTITUTE OF ENGINEERING
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

by
Iqbal A. Goralwalla
June 1992

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Prof. Erol Arkun (Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Assoc. Prof. Abdullah U. Tansel

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Prof. Gültekin Özsoyoğlu

Approved by the Institute of Engineering:

M. Baray

Prof. Mehmet Baray, Director of Institute of Engineering

ABSTRACT

AN IMPLEMENTATION OF A TEMPORAL RELATIONAL DATABASE MANAGEMENT SYSTEM

Iqbal A. Goralwalla
M.S. in Computer Engineering and Information Science
Supervisor: Prof. Erol Arkun
June 1992

In this work, the implementation of a temporal database management system is reported. This system has been implemented on top of an existing database system that manipulates relations with set-valued attributes. The temporal relational model together with the temporal algebra are described. The basic set of operations of the extended relational algebra have been modified to handle temporal attributes. New operations have been added to help in the extraction of information from historical relations. These operations convert one attribute type to another and do selection over the time dimension. Moreover, a statistical interface has been added to the system. This interface includes aggregate functions and a new operation, *enumeration*, which derives a table of uniform data for a set of time points or intervals, from a three dimensional historical relation. A performance evaluation of the system is carried out by executing sample queries against different types of databases: snapshot, snapshot/nested and historical.

Keywords: temporal database, extended relational algebra, enumeration operation, aggregation operation, set-valued relations, performance evaluation.

ÖZET

ZAMAN BOYUTLU İLİŞKİSEL BİR VERİ TABANI YÖNETİM SİSTEMİNİN GERÇEKLEŞTİRİLMESİ

Iqbal A. Goralwalla
Bilgisayar Mühendisliği ve Enformatik Bilimleri Bölümü
Yüksek Lisans
Tez Yöneticisi: Prof. Dr. Erol Arkun
Haziran 1992

Bu çalışmada zaman boyutlu ilişkisel bir veritabanı yönetim sisteminin gerçekleştirilmesi anlatılmaktadır. Bu sistem küme değerli öznitelikleri olan ilişkileri kullanan mevcut bir sistem üzerine kurulmustur. Zaman boyutlu ilişkisel model zaman boyutlu ilişkisel cebir ile birlikte tanımlanmaktadır. Zamansal verilere ulaşabilmek için ilişkisel cebire ait temel işlemler genişletilmiştir. Aynı zamanda geçmişe ait ilişkiselden bilgi çıkarımında yardımcı olması amacı ile yeni işlemler eklenmiştir. Bu işlemler bir öznitelik türünü diğerine çevirmekte ve zaman boyutu üzerinden seçim yapmaktadır. Ayrıca bir istatistik arabirimi de sisteme eklenmiştir. Basit istatistik fonksiyonlar ve zaman aralıklarında seçme işlemi bu arabirimin ana öğeleridir. Zaman aralıklarında seçme işlemi, verilen bir üç boyutlu geçmişe ait ilişkiselden bir zaman noktalar kümesi veya zaman aralıklar kümesi için geçerli bir veri tablosu oluşturmaktadır. Anlık, anlık/yuvalı ve tarihsel olmak üzere üç ayrı veritabanı türünün örnek sorularla sorgulanması ile sistemin başarım değerlendirilmesi yapılmıştır.

Anahtar Kelimeler: zaman boyutlu veritabanı, genişletilmiş ilişkisel cebir, zaman aralıklarında seçme işlemi, basit istatistik işlem, küme değerli bağıntılar, başarım değerlendirmesi.

ACKNOWLEDGEMENT

Heartiest thanks go to my family for their everlasting love, concern and moral support which enabled me to pursue my dream.

I am really fortunate to have Dr. Erol Arkun as my advisor. Not only during the thesis work, but all along my education at Bilkent University, he has always had time to advise and guide me along the right path. I am proud to say that he has played an essential role in shaping my career.

Bulk of the thesis work was done under the supervision of Dr. Abdullah Tansel. I am grateful to him for introducing the interesting subject of Temporal Databases to me. His understanding, dedication, patience and time spent for lengthy discussions over the e-mail was the primary key to the success of the thesis.

I appreciate and thank Dr. Gültekin Özsoyoğlu for his willingness to be on my thesis committee and for his invaluable comments and suggestions.

Special thanks go to Erkan Uçar and his family for their love, kindness and hospitality. They have left a warm place in my heart for the turkish people and culture.

Last but not the least, I would like to thank Erkan Tın for translating the abstract of the thesis to turkish, Bilge Aydin for always saying *yes* to my never ending requests, Corina Stoica for helping in the graphs, and to Prof. Mehmet Baray, the academic staff and research assistants of Computer Engineering and Information Science and the Bilkent Computer Center staff for creating a pleasant and comfortable atmosphere to work in.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1.1 Overview

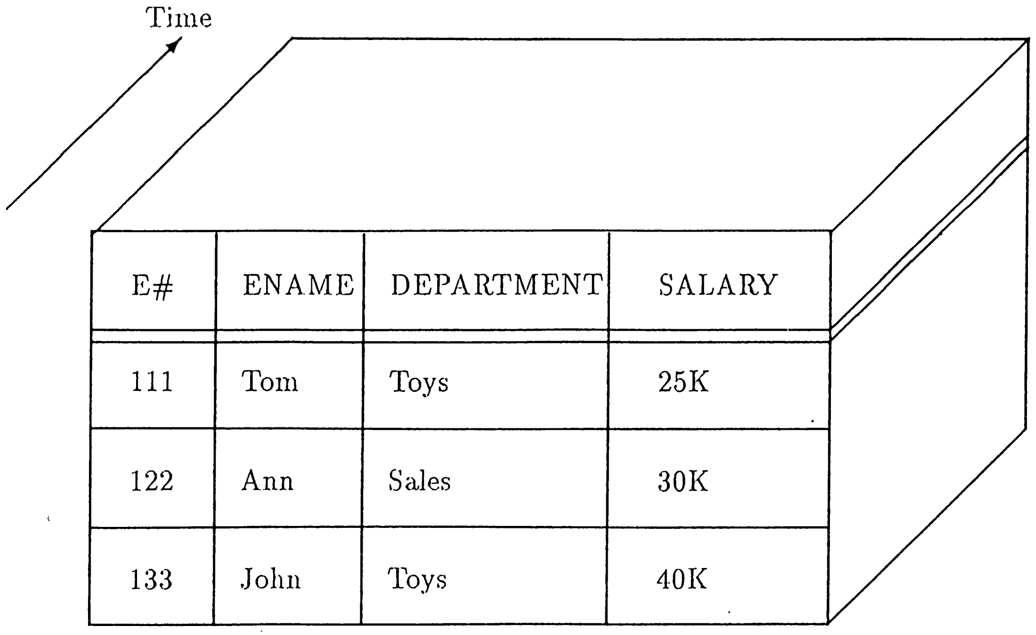
A database contains data pertaining to an organization and its activities. It forms a data repository from which information is extracted for various purposes. Databases in general carry the most recent data.

Time is an attribute of all real-world phenomena. Events occur at specific points in time; objects and the relationships among objects exist over time. The ability to model this temporal dimension of the real world is essential to many information system applications. Examples of these are econometrics, banking, inventory control, medical records, airline reservations, versions in CAD/CAM applications, statistical and scientific data, etc. Yet, none of the three major data models, namely, relational, network, hierarchical supports the time varying aspect of real world phenomena. Conventional databases can be viewed as **snapshot** databases in that they represent the state of an enterprise at one particular time i.e they contain only current data. *Employee* in Figure 1.1 is an example of a relation in a snapshot database. As a database changes, out-of-date information, representing past states of the enterprise, is discarded. However as mentioned above, in many applications there is an obvious need for both current and the past data (possibly future data as well).

The activities of an organization are an on-going process and its information needs and processing capabilities should be considered in a time perspective. That is, to support managerial information needs, as well as others, the database should possess a temporal dimension to store and manipulate time varying data. However, in most models, the time reference of an attribute is carried as another special attribute. This approach is a rather ad hoc and limited solution. It either creates undue data redundancy and/or provides limited time processing capacity.

E#	ENAME	DEPARTMENT	SALARY
111	Tom	Toys	25K
122	Ann	Sales	30K
133	John	Toys	40K

Figure 1.1: EMPLOYEE Relation.



E#	ENAME	DEPARTMENT	SALARY
111	Tom	Toys	25K
122	Ann	Sales	30K
133	John	Toys	40K

Figure 1.2: Generic representation of the EMPLOYEE relation.

The attributes of an object (i.e an entity or a relationship) assume different values over time. The set of these values form the history of that object. A database which maintains past, present and future data (i.e object histories) is called a **Temporal** Database (**TDB**). Figure 1.2 shows the temporal (generic) version of the *Employee* relation in Figure 1.1. Temporal relations can be visualized as a three dimensional cubic structure with time forming the third dimension, the other two being the attribute and tuple dimensions.

Two basic aspects of time are considered in databases which incorporate time. These are the *valid* and *transaction* times. The former denotes the time when an attribute value becomes effective (begins to model reality), while the latter represents the time when a transaction was posted to the database. Usually the valid and transaction times are the same. However, the difference arises when an update to an attribute value is posted to the database at a time which is different than the time when the update became valid. Snodgrass [26] classifies databases into four categories with respect to the valid and transaction times. A *snapshot* database does not have any time dimension. *Rollback* databases are modelled by transaction time. In this case the database can be seen at any time in the past. *Historical* databases have valid time and model the entire history of objects seen as of the present time (*now*). *Temporal* databases carry both valid and transaction times, hence incorporating the features of rollback and historical databases. In this work we use valid time and refer to the term temporal database liberally to stand for a database which carries any kind of time-varying data.

There are two possible directions which can be followed for handling temporal data. One alternative is the development of a new model to support time dimension and the other involves augmenting existing data models to support time dimension in a coherent way. Section 1.2 gives more details regarding these two approaches. The extensions to the relational model to support time fall into two categories, namely,

- *Tuple Time-Stamping*: This extension involves attaching time-stamps to tuples and keeps the relations in First Normal Form (1NF), i.e all the attributes in the relations are atomic. Each relation has two additional attributes, TBEG and TEND to represent the time when an attribute value becomes valid and the time when that attribute value is updated to a new value respectively. In this extension, each relation contains a time varying attribute (or groups of attributes changing at the same time) with a key (E#) as shown in Figure 1.3 (a) and Figure 1.3 (b). Non-time varying attributes are collected into another relation as seen in Figure 1.3 (c).
- *Attribute Time-Stamping*: This approach requires time-stamps to be attached to attributes. Each attribute value is a pair, $< t, v >$ where t is either a temporal set[1] or a time interval[2] and v is an atomic value. $< t, v >$ is called a temporal atom and asserts that the attribute value v is valid during the time periods represented by t. For instance, <[Jan 89, June 90), 32K> is a temporal atom which asserts that the salary value 32K was valid from Jan 89 to June 90. In this case, relations are in Non-First Normal Form (N1NF). Figure 1.4 is an example of attribute time-stamping. Note that in this work, for the sake of simplicity, we use time intervals as time-stamps.

In this work, we extend the relational model and attach time-stamps to attributes as opposed to tuples. We feel this approach is closer to user thought process as compared to tuple time-stamping. In addition, there is minimum data redundancy as all historical data belonging to an object (attribute) is modelled in one single tuple. Arguments in favour of attribute time-stamping are given in [12].

[1]A temporal set is a set of disjoint time intervals. Examples of temporal sets are {[5,10)}, {[5,10), [20,30)}, etc.

[2]A time interval is a pair, [l,u) where l and u represent the lower and upper bounds of the interval respectively.

E#	DEPARTMENT	TBEG	TEND
111	Shoe	Jan 84	Jan 85
111	Toys	Jan 85	now
122	Sales	Jan 86	now
133	Toys	Jan 89	now

(a)

E#	SALARY	TBEG	TEND
111	20K	Jan 84	June 85
111	25K	June 85	now
122	30K	Jan 86	Jan 88
133	32K	Jan 89	June 90
133	40K	June 90	now

(b)

E#	ENAME
111	Tom
122	Anne
133	John

(c)

Figure 1.3: Tuple Time-Stamping

E#	ENAME	DEPARTMENT	SALARY
111	Tom	{<[Jan 84, Jan 85), Shoe>, <[Jan 85, now], Toys>}	{<[Jan 84, June 85), 20K>, <[June 85, now], 25K>}
122	Ann	{<[Jan 86, now], Sales>}	{<[Jan 86, now), 30K>}
133	John	{<[Jan 89,now], Toys>}	{<[Jan 89, June 90), 32K>, <[June 90, now], 40K>}

Figure 1.4: Temporal version of the EMPLOYEE Relation.

1.2 Previous and Related Work

In the last decade, there has been extensive research activity on temporal databases. Bolour, *et al.* contains a comprehensive survey of the role of time in information processing [4]. A bibliography of time in information systems is given in [27, 28]. In addition, Soo [30] gives a bibliography on Temporal Databases while Snodgrass gives a report on the status and research directions in Temporal Databases in [29]. More recently, Mckenzie and Snodgrass [19] survey extensions of relational algebra that can query databases recording time-varying data. They identify criteria for evaluating temporal algebras and evaluate several time-oriented algebras against these criteria.

Development of a new model to handle the temporal aspects of information systems has been reported in [24, 25]. In this model time varying data is visualized as a time sequence collection which is represented as a set of triples (surrogate, time, value). Another model has been proposed by [14] to handle complex objects and their temporal variation. Ginsburg and Tanaka propose a record based system to model histories of financial transactions [11].

An extension to the entity relationship (ER) model for handling time has been proposed by [16]. Elmasri and Wuu [7] also extend the ER model by incorporating the concept of lifespan to entities and relationships.

Most of the research however, has been concentrated in extending the relational model [5] to handle time in an appropriate manner. The extensions can be divided into two main categories. The first approach uses 1NF relations to which special time attributes are added (tuple time-stamping) and object (attribute) histories are modelled by several 1NF tuples [1, 17, 20, 26, 23].

The second approach uses N1NF relations and attaches time to attributes (attribute time-stamping) in which case the object history is modelled by a single N1NF tuple [10, 18, 31, 32, 35].

There have been three recent studies which use attribute time-stamping. Clifford uses time points as time stamps [6] whereas Gadia [10] and Tansel [31] use time intervals as time stamps. Gadia and Tansel's approach both have the same three dimensional view, however they differ in the manner temporal data is refered to. Gadia extracts snapshots from homogeneous[3] temporal relations whereas Tansel either applies algebra operations directly or normalizes the relations before applying the operators. Tansel further explores the structuring of nested historical relations in [35].

[3]A temporal relation is homogeneous if in a tuple, each of its attributes has values over the same time period.

1.3 Scope of the thesis

It is obvious that substantial research activity on temporal databases has been carried out. Modelling temporal data and query languages were the topics mostly investigated. However, there has been very little work in the implementation of temporal databases. TQUEL [26] is the only exception which is a prototype implementation on INGRES database system. The underlying model is based on attaching time stamps to tuples, hence 1NF relations.

In this work we report the implementation of a temporal relational database management system (TDBMS) as proposed in [31]. Our aim is twofold; demonstrating the feasibility of temporal databases with attribute time-stamping and using this implementation to further study various aspects of temporal database management systems, i.e., their performance, features of user friendly query languages, estimators for overhead caused by temporal dimension, etc. We use an existing relational database system, ERAM as described in Chapter 2, supporting one level of nesting [21, 22] and implement a temporal relational database on top of it. We also extend the existing relational operators in this system and introduce new operators to handle temporal attributes. Moreover, we add aggregate functions to handle temporal attributes and implement the *Enumeration* operation [32] which produces a uniform set of data for a series of time intervals, so that the data can easily be used for statistical analysis. A performance study of the implemented system is also carried out. We investigate the performance of snapshot databases, historical databases and databases that allow set-valued attributes by executing different queries with varying characteristics.

Time is attached to attributes as opposed to adding it to tuples. To the best of our knowledge, this is the first implementation of a temporal relational database which uses nested relations and attribute time-stamping. We believe it will be very useful in testing the practicality of the theory developed so far.

The thesis is organized into six chapters. In Chapter 2, the Extended Relational Database System, ERAM, is described, and its model and relational algebra are outlined. Chapter 3 contains the temporal relational model together with its algebra. In Chapter 4, details of the implementation of the temporal database management system (TDBMS) are given. A performance evaluation of the implemented system is given in Chapter 5. The thesis concludes with Chapter 6.

Chapter 2

THE EXTENDED RELATIONAL MODEL DBMS (ERAM)

2.1 Extended Relational Model

Relations restricted to atomic attributes are not always desirable in application areas such as office information systems, social sciences, medical computing, CAD/CAM, textual data processing, etc. This is because these applications involve complex objects and the first normal form restriction of the relational model makes it difficult to model such objects. For modelling these application areas, attributes whose values are sets of atomic values have been proposed [13, 21, 22]. The extended relational model [21, 22] allows relations with set-valued attributes where there is only one level of nesting. It has specifically been formulated for statistical databases, and hence includes powerful aggregation features.

Let U be the set of all values regarded as atomic such as integers, reals, character strings and the value null. If the domain of an attribute is a subset of P(U), where P(U) is the power set of U, then it is called a *nested* or *set-valued* attribute. Let $D_i \subseteq U$ for i=1,...,n. Formally, a nested relation is a subset of $L_1 \times L_2 \times ... \times L_n$ where L_i is the domain of the i^{th} attribute, for i=1,...,n. Then, $L_i = D_i$ or $L_i = P(D_i)$. Atr(R) represents the set containing attributes of relation R. The degree of a relation is the number of its attributes.

2.2 Extended Relational Algebra

Extended relational algebra (ERA) includes the five basic operations of relational algebra with extensions for handling nested relations, and the **aggregate formation** [15],

aggregation-by-template, **pack** and **unpack** operations [21, 22]. The definitions of Cartesian product and projection operations from relational algebra apply directly to nested relations. For union and difference, in addition to the compatibility of relation schemes, the corresponding attributes in both relations must have the same nesting depth. Their definition is the same as the union and set difference operations of relational algebra. In the case of natural join, the common attributes are either both nested or atomic. Other operations modified or introduced in ERA are :

Selection: This operation has been extended for nested attributes, by introducing set-theoretic conditions. Let R be a relation, A and B be attributes of R, v be in U and F be a formula of the form, $F = A\theta B$ or $A\theta v$ where either both operands of F are atomic or both are nested, then,

$$\theta \in \begin{cases} \{=, <, \leq, >, \geq, \neq\} & \text{if both operands are atomic} \\ \{=, \subset, \subseteq, \supset, \supseteq, \neq\} & \text{if both operands are nested} \end{cases}$$

Aggregate formation [15]: Let R be a relation with attributes Atr(R) and X be a subset of Atr(R) with degree k. Let f be an aggregate function and A be an atomic attribute of R. The relation $R < X, f_A >$, then hasdegree $k+1$. Aggregate formation operator first partitions tuples of relation R such that tuples having the same X component are in the same partition. The function f is then applied to A-component of tuples in each partition. The X-value and the calculated aggregate value form the result for each partition. **Aggregation-by-template** [21, 22] is based on grouping tuples of a relation while the aggregate formation is based on partitioning the tuples, i.e., groups may overlap.

Pack [21, 22]: The pack operation, when applied to attribute A of a relation R, collects the values in attribute A into a single tuple component for tuples whose remaining attributes agree. In case of a set-valued attribute, pack operation combines the sets instead of creating another level of nesting. This operation is similar to the one attribute nest operation of [13, 8].

Unpack [21, 22]: The unpack operation, creates a family of tuples for each tuple of R when it is applied on one of R's set-valued attributes. One tuple is created for each element of the set in the attribute value. If unpack is applied on an atomic attribute, R remains unchanged. This operation is similar to the one attribute unnest operation of [13, 8].

Set-formation [21, 22]: The set-formation operation, when applied to an atomic attribute A of a relation R, replaces the tuple component for A by its singleton set, in each tuple of R. If A is nested, then R remains unchanged.

Formal definitions of these operations, as well as their syntax can be found in [9].

2.3 ERAM System Architecture

ERAM is a database management system which is based on the extended relational model and extended relational algebra. It has been implemented in C- programming.language on top of the UNIX version 4.1 BSD operating system. It consists of four modules, namely :

(a) **The file management system (FMS)** is the heart of the system. It is invoked by a higher level module when required. The FMS performs the functions of reading and writing tuples, and garbage collection in the relation instances.

(b) **The relation maintenance module** retrieves, updates, creates and destroys relations. All these functions are eventually executed as calls to FMS routines.

(c) **The algebra module** has been built on top of the FMS. All algebra operations are eventually executed as calls to FMS routines.

(d) **The command interpreter** is the highest level in the implementation in ERAM. It supports a relationally complete query language and relation maintenance commands.

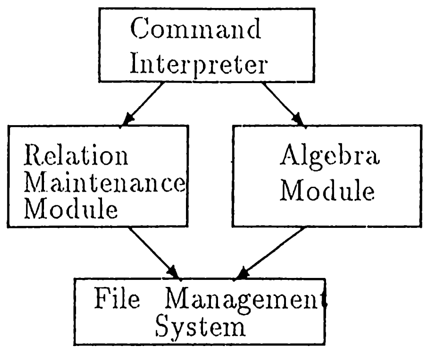


Figure 2.1: System Architecture of ERAM.

The relationships between the above modules is shown in Figure 2.1. Details of the modules and syntax of their commands are given in [9]. We do not include them here to save space. ERAM has been built on top of the Unix internal file structure and Unix standard I/O library. The **relation descriptor**, a core-resident data structure, is the most widely used data structure in the system. Apart from information about the relations in use, it also includes file pointers to the data structures and temporary variables such as the current position of a file, tuple identifier, total number of tuples in the file buffer, and the position of a bucket or a tuple in the file buffer. A relation descriptor is allocated whenever a relation is open and deallocated when the relation is closed. ERAM has *sequential* and *indexed* access methods, the former accesses a relation tuple by tuple while the latter first retrieves the tuple identifier from the primary index file, and then gets the tuple from the relation.

Chapter 3

THE TEMPORAL RELATIONAL DBMS

3.1 The Temporal Relational Model

In our temporal model, each time-varying attribute is assigned a <time, value> pair. The time part of this pair is taken to be the *interval* in which the value is valid as opposed to the *time point* at which the value became valid. The latter approach creates complications in expressing and interpreting the relational algebra operations since it splits the time interval between two successive pairs, causing the successor pair to be examined if the time duration over which the value is valid is needed. Due to this, we opted for the former approach and represent time-varying attributes as triplets of the form, $< [l, u), v >$. [l,u) is the time component (interval), with l and u standing for the lower and upper bounds of the interval respectively. v is the data value which is valid over the time interval [l,u).

T is the set of time points which is a total order under the less than-equal-to ($\leq$) relation. The points are identified relative to an origin t_0, as shown below:

$$T = \{t_0, t_1, ..., t_i, ..., now\}$$

$$t_0 < t_1 ... < t_i < ...now$$

$$t_i = t_{i-1} + 1 \text{ and } t_i = t_0 + i$$

t_0 is the starting time and *now* is the marking symbol for the current time. An interval whose upper bound is *now*, expands as the clock ticks. We do not specify any time unit. It is left to the user. T_I is the set of intervals defined over the time points in T. T_I is a subset of $T \times T$ where $\times$ denotes the Cartesian product:

$$\begin{aligned} T_I \;=\; & \{[l, u) | l < u \wedge t_0 \leq l < now \\ & \wedge t_0 < u \leq now \wedge < l, u > \in T \times T\} \end{aligned}$$

Let $D_{a_1}, \cdots, D_{a_m}$ be subsets of U and $D_{s_1}, \cdots, D_{s_m}$ be subsets of P(U). $D_{t_1}, \cdots, D_{t_m}$ are the subsets of $T_I \times U$ and $P(D_{t_1}), \cdots, P(D_{t_m})$ are their corresponding power sets. Consider a collection of sets $E_1, E_2, \cdots, E_n$ where E_i is one of the above defined sets $D_{a_1}, \cdots, D_{a_m}, D_{s_1}, \cdots, D_{s_m}, D_{t_1}, \cdots, D_{t_m}, P(D_{t_1}), \cdots, P(D_{t_m})$ for i = 1,$\cdots$,n. A ***historical relation*** (HR), defined on the sets $E_1, E_2, \cdots, E_n$, is a subset of the Cartesian product $E_1 \times E_2 \times \cdots \times E_n$. From the definition, it is clear that a HR is a non-first normal form relation whose nesting depth is at most one.

A historical relation may have four types of attributes, namely,

- *Atomic* attributes: contain atomic values, that is, they receive values from domains which are subsets of U.
- *Triplet-valued* attributes: contain triplets as atomic values. A triplet is of the form $< [l, u), v >$, where [l,u) is the time interval of which l is the lower bound and u is the upper bound, and v is the value field. The triplet asserts that the value is valid over the time interval denoted by [l,u).
- *Set-valued* attributes: are sets of atomic values. These values are considered independent of time.
- *Set-triplet-valued* attributes: contain sets of triplets as values. Each set is a collection of one or more triplets, defined over a subset of the interval $[t_0, now]$ and represents the attribute's history. A set-triplet-valued attribute models the history of an attribute of an object.

The following notation is used in naming the attributes of a historical relation. Let A, B, $\cdots$ be attributes of a relation. Atomic attributes are referred to by their names, Set- valued attributes are prefixed by a "*" , e.g. *A, *B, $\cdots$ Triplet-valued attributes are prefixed by a dollar sign, \$A, \$B, $\cdots$, and Set-triplet valued attributes are prefixed by a star and dollar, e.g. *\$A, *\$B, etc. To refer to the components of a triplet-valued attribute, \$A, we use $\$A_l$, $\$A_u$ and $\$A_v$. They refer to the lower bound, upper bound, and value fields of the triplet respectively. If A is an attribute of relation R , C_A denotes the remaining attributes of R, i.e. Atr(R) $-$ $\{A\}$.

3.2 The Temporal Relational Algebra

The set of temporal relational algebra (TRA) operations consists of the five basic operations of relational algebra with extensions for handling temporal relations with one level of nesting. The Pack and Unpack operations are as described in Chapter 2, Section 2.2, revised for temporal relations. The Selection operation is revised so that when a triplet-valued attribute, say \$X, is used, its components can be referenced. That is, $\$X_l, \$X_u$, or $\$X_v$ are allowed in formulas. New operations are added to TRA [31] and they are formally described in the following sections:

3.2.1 Triplet-decomposition (T-DEC)

Let R be a relation of degree n and A be one of its attributes, then, the T-DEC operation breaks a triplet valued attribute \$A into its components. It adds two new attributes to the relation for representing the time interval, one each for the upper and lower bounds. The value field replaces \$A. A new relation with degree $n + 2$ is created.

$$\begin{aligned} TDEC_{\$A}(R) \ = \ & \{t \circ l \circ u | (\exists t')(t' \in R \wedge \\ & t[C_A] = t'[C_{\$A}] \wedge t[A] = t'[\$A_v] \wedge \\ & l = t'[\$A_l] \wedge u = t'[\$A_u]\} \end{aligned}$$

The symbol $\circ$ denotes concatenation. The new attributes are named $\$A_l$ and $\$A_u$. $\$A_l$ is the $(n+1)^{st}$ and $\$A_u$ is the $(n+2)^{nd}$ attribute.

3.2.2 Triplet-formation (T-FORM)

T-FORM creates a triplet-valued attribute from the three atomic attributes L, U, and V which correspond to the lower bound, upper bound, and value components of the triplet respectively. The former two attributes are used to form the time interval over which the value is valid. The resulting triplet-valued attribute, $\$V$, replaces attribute V and the other two attributes are projected out. Let R be a relation with degree $n + 2$, L, U, $V \in$ Atr(R), and $C_F =$ Atr(R) $- \{L, U, V\}$.

$$\begin{aligned} TFORM_{V,L,U}(R) \ = \ & \{t | (\exists t')(t' \in R \wedge t[C_F] = t'[C_F] \wedge \\ & t[\$V_v] = t'[V] \wedge t[\$V_l] = t'[L] \wedge t[\$V_u] = t'[U])\} \end{aligned}$$

The resulting relation has degree n. Applying triplet-formation to the attributes created by a triplet-decomposition operation produces back the original relation, that is,

$TFORM_{A,A_l,A_u}[TDEC_{\$A}(R)] = R.$

3.2.3 Slice (SLICE)

Let R be a relation and *$A* and *$B* be two triplet-valued attributes. $Slice_{\$A,\$B}(R)$ aligns the time of *$A* with respect to the time of *$B*.

$$\begin{aligned}
SLICE_{\$A,\$B}(R) \;=\; & \{t|(\exists t')(t' \in R \wedge t[C_{\$A}] = t'[C_{\$A}] \wedge \\
& t[\$A_v] = t'[\$A_v] \wedge t[\$A_l] \geq t'[\$B_l] \wedge \\
& t[\$A_u] \leq t'[\$B_u] \wedge t[\$A_l] \geq t'[\$A_l] \wedge \\
& t[\$A_u] \leq t'[\$A_u] \wedge \\
& (t[\$A_l] = t'[\$A_l] \vee t[\$A_l] = t'[\$B_l]) \wedge \\
& (t[\$A_u] = t'[\$A_u] \vee t[\$A_u] = t'[\$B_u])\}
\end{aligned}$$

In other words, the intersection of intervals of *$A* and *$B* is assigned to *$A* as its time reference, if an intersection is found. The new triplet receives its data value from attribute *$A*. If no intersection is found, the tuple is simply discarded. The Slice operation implements the *when* clause of English sentences. Two other versions of the Slice operation can also be defined, based on set union and set difference. These make the manipulation of intervals easier.

(a) **Union Slice (USLICE)**: Let R be a relation and *$A* and *$B* be two triplet-valued attributes in Atr(R). USLICE operation creates a new relation whose attributes are Atr(R) $-$ {*$A*} $\cup$ {**$A*}.

$$\begin{aligned}
USLICE_{\$A,\$B}(R) \;=\; & \{t|(\exists t')(t' \in R \wedge t[C_{\$A}] = t'[C_{\$A}] \wedge \\
& x \in t[*\$A] \wedge x_v = t'[\$A_v] \wedge (x_l \leq t'[\$A_l] \wedge \\
& x_u \geq t'[\$A_u] \wedge (x_l \leq t'[\$B_l] \wedge x_u \geq t'[\$B_u] \wedge \\
& (x_l = t'[\$A_l] \vee x_l = t'[\$B_l]) \wedge (x_u = t'[\$A_u] \vee \\
& x_u = t'[\$B_u]) \wedge (t'[\$A_u] \geq t'[\$B_l] \vee \\
& t'[\$B_u] \geq t'[\$A_l])) \vee x_l = t'[\$A_l] \wedge x_u = t'[\$A_u]) \vee \\
& (x_l = t'[\$B_l] \wedge x_u = t'[\$B_u] \wedge \\
& (t'[\$A_u] < t'[\$B_l] \vee t'[\$B_u] < t'[\$A_l]))\}
\end{aligned}$$

For each tuple, the time interval component of attribute *$A* is assigned the union of the time intervals of attributes *$A* and *$B*. As opposed to the intersection version of the Slice operation, here we have two cases. If the time intervals of *$A* and *$B* intersect, the result is a single large interval. If there is no intersection, the result has two components, the interval of *$A* and the interval of *$B*, forming two triplets. As a consequence, USLICE operation creates a set triplet-valued attribute from a triplet-valued attribute.

(b) **Difference Slice (DSLICE)**: Let R be a relation and *$A* and *$B* be two triplet-valued attributes in Atr(R). DSLICE operation creates a new relation whose attributes are Atr(R) $-$ {*$A*} $\cup$ {**$A*}.

$$\begin{aligned} DSLICE_{\$A,\$B}(R) \;=\; & \{t|(\exists t')(t' \in R \wedge t[C_{\$A}] = t'[C_{\$A}] \wedge \\ & x \in t[*\$A] \wedge x_v = t'[\$A_v] \wedge (x_l = t'[\$A_l] \wedge \\ & x_u = t'[\$B_l] \wedge t'[\$A_l] < t'[\$B_l] < t'[\$A_u] \leq t'[\$B_u]) \vee \\ & (x_l = t'[\$B_u] \wedge (x_u = t'[\$A_u] \wedge \\ & t'[\$B_l] \leq t'[\$A_l] < t'[\$B_u] < t'[\$A_u]) \vee \\ & (((x_l = t'[\$A_l \wedge x_u = t'[\$B_l]) \vee (x_l = t'[\$B_u] \wedge \\ & x_u = t'[\$A_u])) \wedge t'[\$A_l] < t'[\$B_l] < t'[\$B_u] < t'[\$A_u]) \vee \\ & (x_l = t'[\$A_l] \wedge x_u = t'[\$A_u]))) \wedge \\ & (t'[\$A_u] < t'[\$B_l] \vee t'[\$B_u]) < t'[\$A_l]))\} \end{aligned}$$

For each tuple, the time interval component of attribute *$A* is assigned the difference of the time intervals of attributes *$A* and *$B*. As was the case in USLICE, DSLICE also creates a set triplet-valued attribute from a triplet-valued attribute. If the time interval of *$B* is contained in the time interval of *$A*, the difference results in two triplets.

3.2.4 Drop-time (DROP-TIME)

This operation gets rid of the time components of a triplet-valued or a set-triplet valued attribute and converts it into an atomic or a set-valued attribute, respectively. Let R be a relation and $A \in$ Atr(R).

$$DROPTIME_A(R) = \begin{cases} \{t|(\exists t')(t' \in R \wedge t[C_{\$A}] = t'[C_{\$A}] \wedge t[A] = t'[\$A_v])\} \\ \text{if } A \text{ is a triplet-valued attribute} \\ \\ \{t|(\exists t')t' \in R \wedge t[C_{\$A}] = t'[C_{\$A}] \wedge x \in t[*\$A] \wedge x_v \in t[A])\} \\ \text{if } A \text{ is a set triplet-valued attribute} \\ \\ R \text{ otherwise} \end{cases}$$

Note that the degree of the resulting relation is the same as that of R. Drop-time allows us to manipulate static (without time) part of temporal relations.

3.2.5 Enumeration (ENUM)

The enumeration operation [32] derives a table of uniform data, for a set of specified time points or intervals, from a three dimensional historical relation. Together with aggregate functions, the enumeration operation provides a statistical interface which enables transformations of data into tabular form suitable for statistical analysis. Three issues arise when applying aggregates to historical data, namely,

(a) Aligning the time reference of attributes involved in aggregation operations, so that each attribute has the same time reference

(b) Selecting attribute values at specified time points before applying aggregate functions.

(c) Time units of the time frame in which data is viewed and time granularity used to model the database are not the same, i.e., yearly salary is required from a historical database in which the time unit is a month.

To handle case (a), unpack and slice operations can be used. To deal with the issues of case (b) and (c), two versions of the enumeration operation have been developed, and are described in the following two sections. Implementation details are given in Chapter 4.

3.2.5.1 Enumeration: First Version (ENUM1)

This version of the enumeration operation returns the values of the designated attributes at the specified time points. Let:

- R be a historical relation,
- $X \subseteq \text{Atr}(R)$, where X_i, $1 \leq \text{i} \leq |X|$, is an attribute, $|X|$ denotes the number of attributes in X,
- T be a single column relation whose tuples, $\{t_1, t_2, \cdots, t_n\}$ are the specified time points,
- X_s be the atomic and set-valued attributes in X,
- X_t be the triplet-valued attributes in X,
- X_{st} be the set triplet-valued attributes in X, i.e., $X = X_s \cup X_t \cup X_{st}$.

ENUM1 can then be defined as,

$$\begin{aligned} R < X > T \;=\; & \{s \circ t | (\exists s')(s' \in R \wedge t \in T \wedge \\ & s[X_s] = s'[X_s] \wedge (s[Y] = s'[\$Y_v] \wedge \\ & s'[\$Y_l] \leq t < s'[\$Y_u] \text{ for } \$Y \in X_t) \wedge \\ & (s[Z] = s'[\$z_v] \wedge \$z_l \leq t < \$z_u \\ & \text{for all } \$z \in s[\$Z], \text{ for } \$Z \in X_{st}))\} \end{aligned}$$

$R{<}X{>}T$ creates a new relation whose degree is $|X| + 1$. The remaining attributes, i.e those not in X are discarded. It should be noted that the definition of $R{<}X{>}T$ allows set-valued attributes in the result. However, these can easily be unpacked before applying a statistical function.

3.2.5.2 Enumeration: Second Version (ENUM2)

Aggregation operations give rise to semantic issues when applied on historical data. Consider the case when a database is modelled with a time granularity of months. Any reference with respect to days or weeks can unambiguously be resolved since a value which is valid for a month is obviously valid for any day or week of that month. However, when a reference is made with respect to a time unit which is larger than a month, e.g annual salary, we have problems. If there is only one salary value throughout the year, no problem occurs. The ambiguity arises when there are two or more salary values or there are salary values for some months while for the other months they are missing. Such references need some rules of interpretation so that they can unambiguously be resolved. This version of the enumeration operation provides a solution to this problem. It determines an appropriate value for an attribute by utilizing all of its values which are valid over the respective time interval. This value can then be used in statistical analysis. Let:

- R be a historical relation,
- $X \subseteq \text{Atr}(R)$, where X_i, $1 \leq i \leq |X|$, is an attribute,
- $T(\$B)$ be a single column relation whose values, $\{< l_1, u_1 >, < l_2, u_2 >, \cdots, < l_n, u_n >\}$ are the specified time intervals,
- X_s be the atomic and set-valued attributes in X,
- $X_t = \{A_1, A_2, \cdots, A_k\}$ be the triplet-valued and set triplet-valued attributes in X, i.e., $X = X_s \cup X_t$.

ENUM2 can then be defined as,

$$R < X_s, f_1(A_1), f_2(A_2), \cdots, f_k(A_k) > T \;=\; \{s \circ y_1 \circ \cdots \circ y_k \circ \$b | (\exists s')(s' \in R \wedge t \in T \wedge s = s'[X_s] \wedge \$b = t[\$B] \wedge y_i = f_i(\{w | w = \$z_v \wedge (\$z \in s'[A_i] \vee \$z = s'[A_i]) \wedge [\$z_l, \$z_u) \cap [\$b_l, \$b_u) \neq \emptyset\}) \text{ for } i = 1, 2, \cdots, k)\}.$$

Here, $f_1, \cdots, f_k$ are aggregation functions. Each of these apply an aggregation operation on the values of its operand attribute in the specified interval to return a single value as its result. A rich set of aggregation functions has been provided to handle the various subtleties of time. These include:

MIN	Returns the minimum of a set of values.
PMIN	Returns the partial minimum of set of values depending on the duration of validity of the values. In other words, values are adjusted by their validity period.
MAX	Returns the maximum of a set of values.
PMAX	Returns the partial maximum of set of values depending on the duration of validity of the values.
COUNT	Returns the count of a set of values.
SUM	Returns the sum of a set of values.
PSUM	Returns a proportional sum adjusted by the duration of validity of the value. For example if salary is an annual figure, PSUM will return half the salary for a value valid for six months.
AVG	Returns the average of a set of values.
WAVG	Returns the weighted average of a set of values, duration of each value serving as it weight.
FIRST	: Returns the first of values in chronological order.
LAST	: Returns the last of values in chronological order.
LEN	: Returns the length of the time interval of a triplet.

Note that the aggregate formation of the temporal relational algebra is similar to the aggregate formation operation of ERAM. Aggregate functions listed above (apart from *pmax*, *pmin*, *psum* and *wavg*) can also be used in the aggregate formation operation of TRA.

Example queries are given in Chapter 5, Section 5.3 and the corresponding TRA expressions are provided in Appendix D.

Chapter 4

IMPLEMENTATION OF THE TEMPORAL DBMS

4.1 Implementation of ERAM

In this section, we briefly summarize the implementation of ERAM [9]. The data dictionary consists of two files, *direct* and *users*. The USERS relation contains information about all users who are allowed to access the database. The *users* file is just a text file, one line for each user. The *direct* file has a tree like structure and contains two system relations, RELATIONS relation and ATTRIBUTES relation.

The RELATIONS relation contains one tuple for each relation in the database. Some of the attributes of the RELATIONS relation are, Relation name, Tuple size, Owner-id, Keyno1 to Keyno6 (at most six fields comprise the key), Number of tuples, and Atr-pointer (pointer to first page of ATTRIBUTES relation).

The ATTRIBUTES relation contains information about attributes and their domains, one tuple for each attribute. Some of the attributes of the ATTRIBUTES relation are, Attribute name, Attribute number, Attribute type (atomic or nested), Data type (character string, integer or real), Bucket size (for nested attributes), and Attribute size.

The dictionary is organized as a tree structure. At the top level (root), there is an entry for each database. The entry contains only the database name and a pointer to the beginning of the related RELATIONS relation (node). Each entry (tuple) in the RELATIONS relation represents a relation and has a pointer, called *atr-pointer* to the beginning of the related ATTRIBUTES relation (leaf node). When the DBMS is invoked for manipulating a database, the pointer to its RELATIONS relation is read into memory. When a relation is opened, first, the RELATIONS relation is searched for the desired tuple. If found, the system locates the ATTRIBUTES relation by following its *atr-pointer*.

4.2 Implementation of the Temporal Database Management System: Two different approaches

ERAM has successfully been used in several universities in the last five years. Therefore it is a proven system. At the same time, it provides a natural environment for the implementation of temporal databases since it supports nested relations. Our main intention is to implement the TRA on top of ERAM. This requires mapping the temporal attributes of a temporal relation to the corresponding attributes of ERAM. A conversion method is needed to translate the triplet-valued and set-triplet valued attributes of a temporal relation to corresponding attributes of ERAM. This allows using the functionalities of ERAM to model and to implement the TRA operations. Two alternatives are considered, which are:

(i) In this case, the triplet-valued attributes are converted into *three* different *atomic* attributes, that is, $< [l, u), v >$ become l, u and v. For the set-triplet valued attributes, we create three set-valued attributes. This is exemplified by the following example where a set of triplets say attribute *$X, are broken into their components:

$$\{< 1,2,a >, < 3,4,b >, < 5,6,c >\}$$

$$\downarrow$$

*A	*B	*C
$\{1,3,5\}$	$\{2,4,6\}$	$\{a,b,c\}$

The lower bound values (l) of each triplet in the set-triplet valued attribute are combined into a single set-valued attribute. The same is the case for the upper bound (u) and the value (v) fields of each triplet. In other words, for each set-triplet valued attribute consisting of n triplets, we come up with three set-valued attributes each having n elements. It can be seen that each of the set-valued attributes has components which have the same type which is also required by ERAM. Now, these three attributes can easily be handled in ERAM to simulate a set-triplet valued attribute. However, the problem in this alternative is to maintain the order of the components in each set-valued attribute. Otherwise it would be impossible to reconstruct the original triplets from these three separate attributes. Furthermore, duplicates are not allowed in set-valued attributes. Duplicate values may appear when a set-triplet valued attribute is broken into its components. This necessitates conversion of duplicates into unique values so that they can be safely manipulated by ERAM.

(ii) In the second alternative, the triplet-valued attribute are converted to a single atomic attribute of ERAM. That is, $< [l, u), v >$ becomes *luv*, a single value, hence, a single attribute in ERAM. For the set-triplet valued attributes, each of its triplets is combined into one string, *luv*. As an example, $\{< 1,2,a >, < 3,4,b >, < 5,6,c >\}$, becomes 12a, 34b, 56c, a single set-valued attribute in ERAM.

It was noted that in ERAM, instances of a set-valued attribute are kept sorted and without duplicates. This would create a problem for method (i) above since duplicates may appear in the converted result. A solution to this problem would be to attach prefixes to the attribute components, to make them unique. But, this would lead to considerable computational complexity in manipulating and updating the prefixes in each relational operation. Hence, the second method, converting triplets into strings is preffered and is used in the implementation.

4.3 Developing the chosen alternative

4.3.1 Data Formats for Time

We use the following data structure in representing a triplet:

lower bound (l)	upper bound (u)	value (v)

where the lower and upper bounds are represented as:

dd (5 bits)	mm (4 bits)	yy (7 bits)

A time point is represented as *ddmmyy* where *dd* refers to the day, *mm* refers to the month, and *yy* stands for the year. Each of these are specified by only two digits. Hence, the lower and upper bounds are compressed into a total of four bytes. Thus, the system can accomodate 128 years where the time granularity is a day. The value of a triplet is concatenated to these four bytes to form the final string. When needed, they are uncompressed to their original form, in presenting the results and in implementing TRA operations.

4.3.2 Implementation of the Relational Commands

The existing commands of ERAM have been modified to accommodate TRA operations. Changes have been made to the system at appropriate places, either by modifying the code or writing new routines. Since ERAM is a large system involving about 10,000 lines of C code, it was quite difficult to make modifications. Changes were required especially since many routines are used in several places. Additionally, as most of the operations involve writing relations to files and sorting them, care had to be taken to uncompress the temporal attributes before the sorting procedure was carried out so that the required results could correctly be obtained. We now describe all the commands in detail. The details of the command syntax are given in Appendix A. More details on the temporal commands can be found in Appendix B, while details of the other commands are given in [9].

4.3.2.1 System Level Operations

Information about each user has to be entered into the *users* file by the superuser before the user can start using the database management system. The following are the three system level commands.

- CREATEDB : Creation of a database
  EXAMPLE : createdb emp

  The invoker of this command automatically becomes the database administrator (DBA) of the database he/she creates and has the ability to create or destroy any relation in his/her database. The *direct* file, if it does not exist, is also automatically created by this command.

- DELETEDB : Removal of a database
  EXAMPLE : deletedb emp

  This command has to be invoked by the DBA. It destroys the database by removing all the relations, index files and references.

- TDBMS : Invocation of the DBMS
  EXAMPLE : tdbms emp

  This command logs the user to the named database. The system prompt, @, is seen if the user has access rights to the database. At this juncture, the dictionary is opened, variables are initialized and space is allocated. The user can then issue any of the relational commands (accept the system level commands) within the invoked database. To exit from the system, the user simply has to type a *q* at the prompt, thereby closing the system relations, deallocating space and returning control back to the Unix system.

4.3.2.2 Relational Maintenance Commands

These commands are used for the creation, maintenance, and removal of relations from the active database.

- CREATE : Creation of a relation scheme
  EXAMPLE : create employee(e#:=i2 ename:=c15 department:=c10s2 salary:=i2s2)

  A relation scheme, employee, is created with attributes e#, ename, department, and salary. e# and ename are atomic attributes of 2 byte integer, and character string of size 15 respectively. Department and salary are set-triplet valued attributes (indicated by the *s*) of character string and 2-byte integer respectively. They both have a bucket size of 2. A nested attribute is specified in a similar manner as the set-triplet valued attribute except the *s* in the format of the latter is replaced by a *n*. A triplet valued attribute is specified by appending a *t* to the format of an atomic attribute. The data types available for the attributes are character strings (at most 255 characters), integers (2 or 4-byte), and reals (8-byte).

- DESTROY : Removal of a relation
  EXAMPLE : destroy employee

  The DBA may destroy any relation in the database which he/she owns, while a user may destroy only the relations that he owns. The command causes the removal of the relation and primary index files from the Unix system and the tuple for the destroyed relation in the RELATIONS relation is marked as *deleted.*

- COPY : Move tuples between a file and a relation
  EXAMPLE : copy employee() from /home/usr3/iqbal/data
  copy employee(e#,ename) into home/usr3/iqbal/data

  Each line in the Unix file correspond to a tuple in the relation. Attributes in the file are separated by a ";" while instances of a nested attribute and set-triplet valued attribute are separated by a ",". Data in the file is checked for its format, unmatched data on data type or attribute type (atomic/nested) is sent to an error file. The instances of set-valued and set-triplet valued attributes are kept sorted and without duplicates. Similarly, no duplicate tuples are allowed in a relation. Input tuples do not have to contain values of all attributes. Default *null* values are assigned to the attributes not specified. The default values are, the minimum possible values for a 2-byte integer, a 4-byte integer and a real. A null character string is used for a data type of character string.

  Triplet-valued attributes are specified in the file as a character string of 12 characters (4 characters are used for the day, month, and year of the lower and upper bounds) followed by the value component. For example, the triplet <[01,Jan 89, 01,June

90), 32K> would be specified as, 01018901069032. Set-triplet valued attributes are specified similarly, as a set of triplets.

- APPEND : Appending new 1tuples to a relation from a terminal
  EXAMPLE : append employee(e#:=122 ename:="Ann"
  department:={"010186111111Sales"}
  salary:={"01018611111130"})

  The input device for the append operation is a terminal. This command is similar to the *copy from* ··· command and is suitable when a small number of tuples have to be added to a relation. Attributes need not appear in order and as in the *copy* command, null values are assigned to the attributes which are not present. In this example, two triplets are specified. One of them is 010186111111Sales where 010186 stands for the lower bound (Jan 1st,1986), 111111 stands for the upper bound (*now*), and the value field is "Sales." The time point *now* is represented as 111111, differentiating it from the other time points. Ofcourse, in comparisons, the value of *now* is changed appropriately to stand for the largest time value. Instances of set-triplet valued attributes are these triplets separated by a ",".

- DELETE : Deletion of tuples
  EXAMPLE : delete employee where ename="Ann"
  delete employee where departmentv="Sales" (assuming employee has been *unpacked* on the department attribute. See the UNPACK command in the next section.)

  Tuples are selected by specifying a set of conditions. Each condition is a set of expressions of the form *atr-name comp-op atr-name/constant* connected by the logical operators *and* or *or*. *comp-op* is a comparison operator.

  Deletion can also be carried out by specifying any component of a triplet-valued attribute in the conditional expression, i.e., *atr-name*$_l$, *atr-name*$_u$, or *atr-name*$_v$. A set-triplet valued attribute has to be *unpacked* into the corresponding triplet-valued attribute before the deletion command can be applied to any of its components.

- UPDATE : Modification of attribute values
  EXAMPLE : update employee (ename:="Anne") where e#=122
  update employee (salaryv:="35") where departmentv="Sales" (assuming employee has been *unpacked* on the department and salary attributes. See the UNPACK command in the next section.)

  As was the case with the *delete* command, tuples are selected by specifying a set of conditions. Selection of tuples can also be done by specifying any component of a triplet-valued attribute in the conditional expression. Similarly, any component of the triplet-valued attribute can also be modified.

- PRINT : Printing a relation
  EXAMPLE : print employee

  This command displays a relation or the result of an algebra expression on the terminal (default) or sends the output to the printer if the -l option is specified. The printing of temporal attributes is handled appropriately as shown below :

Attribute	Printed
Triplet :	
01018611111130	$< [01/01/86, now], 30 >$
Set-triplet :	
{0101840101855Shoe, 010185111111Toys}	$\{< [01/01/84, 01/01/85), Shoe >,$ $< [01/01/85, now], Toys >\}$

- SCHEME : Printing relation schemes
  EXAMPLE : scheme employee

  This command prints the scheme of a single relation, a set of specified relations, or all the relations in the database (if the relation name is not specified). As was the case for the *print* command, the output device may be a terminal or a line printer (in which case the -l option has to be specified).

- RENAME : Modification of attribute names
  EXAMPLE : rename employee(department:=dept)

  Attribute names can only be modified by the relation owner and the DBA.

4.3.2.3 Algebra Operations

Any algebra expression can be preceded by *"rel-name:="* to store its result as a permanent relation, else the result is displayed on the terminal. The algebra operations available in ERAM are now described briefly. Details are given in [9]. Modifications for handling temporal attributes are pointed out where appropriate.

- UNION : Union of two relations
  EXAMPLE : employee union employee1

  UNION requires the two relations to be compatible, in that the corresponding attributes in the two relations should have the same data type and nesting depth.

- DIFFERENCE : Difference of two relations
  EXAMPLE : employee difference employee1

  As was the case for the *union* command, the two relations have to be compatible. Tuples in the first relation but not in the second are selected as the result.

- CPROD : Cartesian product of two relations
  EXAMPLE : employee cprod employee1

  No index method is provided for this command. For each tuple in the first relation, all the tuples in the second relation are read sequentially. A *$* is used to replace the first character in the second occurrence of the identical name, in case the two relations have any attribute name in common.

- PROJECT : Remove some attributes from a relation
  EXAMPLE : project employee on e# salary

  This operation picks out values of the named attributes. Output tuples are sorted and without any duplicates.

- SELECT : Select tuples on specified condition
  EXAMPLE : select e# salary from employee where departmentv = "Sales"
  (assuming employee has been *unpacked* on the department attribute. See the UNPACK command.)

  The select operation outputs those tuples which satisfy the desired conditions in the selection formula. For triplet-valued attributes, we can specify the components of a triplet in the conditional expression (see the *delete* command in the previous section). In referencing these components, first the compressed string is decomposed into its components. Then, the condition is tested to determine whether the tuple qualifies or not. In addition, the set-theoretic conditions mentioned in Chapter 2, Section 2.2 have been appropriately modified to select desired triplets from a set-triplet valued attribute.

- SETF Set formation
  EXAMPLE : setf employee on ename

  This command changes the specified atomic attribute to a nested attribute. If the specified attribute is nested, the input relation is returned unchanged.

- UNPACK : Unpack a relation on an attribute
  EXAMPLE : unpack employee on salary

  The output relation has the same set of attributes as the original relation, except that the specified attribute is changed to an atomic attribute if it was a nested attribute or a triplet-valued attribute if it was a set-triplet valued attribute. Each tuple of the original relation is output as many times as the number of instances of the specified attribute. If the specified attribute is atomic, the input relation is returned unchanged.

- PACK : Pack tuples on an attribute
  EXAMPLE : pack (unpack employee on salary) on salary

The pack operation, when applied to attribute A of a relation R, collects the values in attribute A into a single tuple component for tuples whose remaining attributes agree. If the specified attribute is atomic but not temporal, the resulting attribute is set-valued, else if the specified attribute is atomic and temporal (i.e triplet-valued), the resulting attribute is set-triplet valued. In case of a set-valued attribute, pack operation combines the sets instead of creating another level of nesting.

- $\underline{R < \{X\} f_A >}$: Aggregate formation
  EXAMPLE : employee<{e# department} sum(salary)>

  Details of this command are given in Chapter 2, Section 2.2. The aggregation attributes have to be atomic. If the specified aggregation attribute is temporal (i.e triplet-valued), aggregation is applied on it's value component. Moreover, if one of the grouping attributes is temporal, then exact equality of tuples has been used to group the tuples. Aggregate functions available are: min, max, sum, avg, count, median, first and last.

  Aggregation-by-template, $R < \{X\}\{Y\} f_A > Z$, is similar to the aggregate formation command except it is based on grouping tuples of a relation while the aggregate formation is based on partitioning the tuples, i.e., groups may overlap.

- NJOIN : Natural join of two relations
  EXAMPLE : employee njoin employeel

  This operation concatenates (not including the join attributes of the second relation) two tuples, which have the same value for the join attributes, and outputs the resulting tuple. If there are no attribute names in common, the *cprod* command is invoked. The NJOIN operation has been modified for the join between triplet-valued attributes and between set-triplet valued attributes. These attributes are first uncompressed into their original strings, written to a file with the other join attributes, checked for equality and finally compressed and stored in a new relation if a match is found.

4.3.2.4 Temporal Algebra Operations

New code has been written for the temporal algebra operations to handle the presence of new attribute types involving the time dimension. These operations have been introduced to manipulate temporal relations by converting one type of attribute to another or forming slices of relations.

- T-DEC : Decomposes a triplet-valued attribute
  EXAMPLE : tdec (unpack employee on salary) on salary

T-DEC breaks a triplet-valued attribute into its lower bound, upper bound and value components forming two additional attributes. An attribute value is first decoded into its components and then each is assigned to its corresponding attribute.

- T-FORM : Formation of a triplet-valued attribute
  EXAMPLE : tform (tdec (unpack employee on salary) on salary) on salary$l$ salary$u$ salary$v$

  T-FORM forms a triplet-valued attribute from the three attributes, salary$l$, salary$u$ and salary$v$ which correspond to the lower bound, upper bound, and the data value component of the formed triplet.

- SLICE : Slices the time component of an attribute
  EXAMPLE : slice (unpack (unpack employee on department) on salary) department by salary

  This operation essentially forms a subset of a relation along the time dimension, in that it checks for the *intersection* of the time components of two triplet-valued attributes. In the above example, if the time intervals of department and salary overlap, the intersection time is assigned to the time component of department. The compressed values of attributes, department and salary are first decoded (after they are unpacked). Then the intersection of their time intervals is calculated. The new value of attribute department is formed and compressed.

- USLICE : Union slices the time component of an attribute
  EXAMPLE : uslice (unpack (unpack employee on department) on salary) department by salary

  This operation is similar to *slice*, except the *union* of the time intervals of department and salary is assigned to the time component of department. As explained in Chapter 3, Section 3.2.3, department becomes a set-triplet valued attribute after this operation.

- DSLICE : Difference slices the time component of an attribute
  EXAMPLE : dslice (unpack (unpack employee on department) on salary) department by salary

  This operation is similar to *slice*, except the *difference* of the time intervals of department and salary is assigned to the time component of department. As explained in Chapter 3, Section 3.2.3, department becomes a set-triplet valued attribute after this operation.

- DROPTIME : Drops the time component of an attribute
  EXAMPLE : droptime employee on department

DROPTIME gets rid of the time component of a temporal attribute, converting it to its snapshot counterpart. In the above example, department becomes a set-valued attribute after the operation.

- ENUM1 : Returns attribute values at specified time points
  EXAMPLE : enum1 employee<{e#} {department salary}> {"010485" "010190"}

  ENUM1 returns the values of designated attributes at specified time points. In the above example, for each of the specified *time points*, each tuple of the employee relation is retrieved. The triplets of the department and salary attributes are examined in turn for intersection with the specified time point. If an intersection is found, the data values of e#, department, and salary are retrieved. The algorithm for implementing ENUM1 is given in Figure 4.1.

- ENUM2 : Returns aggregated attribute values at specified time intervals
  EXAMPLE : enum2 employee<{e#} first(department) psum(salary)>
  {"010185010186" "010190010191"}

  In this version of the enumeration operation, an aggregation function is applied on the values of its operand attribute in the specified interval to return a single value as its result. In the above example, for each of the specified *time intervals*, the triplets of the department and salary attributes are examined in turn for intersection. If an intersection is found, the first department in chronological order from the set of values that qualified is selected, and a partial sum on the set of qualified salary values is performed. The algorithm for implementing ENUM1 is given in Figure 4.2.

```
Procedure Enum1($R < X_s, X_t > T$)
begin                                                  /* Enum1 */
    Retrieve all tuples of T in ascending order
    for each tuple $r$ of $R$ do
        for each attribute $X_i$ in $r[X_t]$ do        /* for each temporal attribute */
            for each triplet $\$x$ in $r[X_i]$ and $t$ in $T$ in chronological order do
                                                       /* where $t$ is a time point in $T$ */
                if $\$x_l \leq t < \$x_v$ then         /* check for intersection */
                    Select this triplet
                else
                    Skip this triplet
            end
        end
        Assemble a result tuple and output it
    end
end                                                    /* Enum */
```

Figure 4.1: Algorithm for ENUM1

```
Procedure Enum2($R < X_s, f_1(A_1), \cdots, f_k(A_k) > T$)
begin                                                  /* Enum2 */
    Retrieve all tuples of $T$ in ascending order
    for each tuple $r$ of $R$ do
        for each attribute $X_i$ in $r[X_t]$ do        /* for each temporal attribute */
            Initialize a structure with $f_i(\emptyset)$
            for each triplet $\$x$ in $r[X_i]$ and $\$t$ in $T$ in chronological order do
                                                       /* where $\$t$ is a time interval in $T$ */
                if $[\$x_l, \$x_v) \cap [\$t_l, \$t_u) \neq \emptyset$ then    /* check for intersection */
                    Apply $f_i(x_v)$
                else
                    Skip this triplet
            end
        end
        Assemble a result tuple and output it
    end
end                                                    /* Enum2 */
```

Figure 4.2: Algorithm for ENUM2

Chapter 5

PERFORMANCE EVALUATION OF TDBMS

5.1 Introduction

Considering the wide variety of temporal queries, large volume of historical data and access methods, performance of temporal queries becomes a critical issue. However, this issue has not received much attention over the past decade. In [2], Ahn and Snodgrass, run a benchmark set of queries to study the performance of their prototype system on four types of databases: static, rollback, historical and temporal. Furthermore, they follow the analytic approach by proposing a model which analyses the input and output cost of temporal queries on various access methods [3]. However, obtaining useful results from an analytic approach is difficult because the actual cost depends on the particular implementation of the TDBMS. In contrast, we follow an empirical approach and measure the actual performance of representative temporal queries on our system. We believe the results will provide useful insight into the performance of historical databases using attribute time-stamping, as well as design of such databases.

Our aim is to measure the performance of our system using different queries in terms of processing time. Cpu-time for each query is collected and calculated by using the UNIX system call, *clock*. We have chosen a set of sample queries with varying characteristics, comparison between atomic attributes, atomic and triplet-valued attributes and triplet-valued and triplet-valued attributes. A list of these queries is provided in Section 5.3. These queries are executed against the following database types:

- **Historical** – **H**: TDBMS (A historical database)
- **Snapshot** – S: Conventional database carrying current data

- **Snapshot/Nested – S/N**: Database which involves set-valued attributes

To insure consistency, these three types of databases contain the same data. The historical database contains past and present data whereas the snapshot database includes only the current data. The snapshot/nested database is created by discarding the time reference of the historical data. We formulated the sample queries by considering the semantics of these databases. To develop a comprehensive idea on the performance of different database types, we make the following comparisons:

- **H:S** – gives an idea on how much performance degradation occurs in moving from snapshot to historical databases.
- **H:S/N** – gives a measure of the overhead introduced by adding time to databases having relations which support set-valued attributes.
- **S:S/N** – reflects the performance change in moving from snapshot databases to databases which support set-valued attributes.

5.2 Generation of Data

5.2.1 Database Schema

A test database, called, *eval* has been created for the evaluation of the database types mentioned in the previous section. It consists of four relations, namely, *Emp*, *Dept*, *Proj* and *Assigned*. These relations are:

(a) **Emp**(ssno, name, address, *$salary, *$skills, *$dname, *$manager)

(b) **Dept**(dno, dname, *$budget, *$manager)

(c) **Proj**(pno, pname, *$budget)

(d) **Assigned**(ssno, pno, *$type of work, *$rating)

In the above schemas, a **$* indicates that the attribute is set-triplet valued, representing historical data. The rest of the attributes are atomic. The *Assigned* relation shows employees and the projects they are assigned to. Attribute *type of work* denotes the different types of work carried out by the employee. For each assignment, the employee gets a rating, say, 0–100 for different periods of time. This is represented by the attribute *rating*. The rest of the relations and their attributes are self-explanatory. We use number of tuples in a relation and the number of triplets in a set-triplet valued attribute as parameters of this database.

5.2.2 Population of Data

Each of the relations mentioned in the previous section, is populated with data generated randomly from an uniform distribution. The random function, *drand48()* which returns non-negative double-precision floating-point values uniformly distributed over the interval [0.0, 1.0), is used to generate random data. The random number is then converted to an attribute value. The range of the attribute values is listed in Figure 5.1. The randomly created data is first written to text files from which it is copied to the respective relations using the *copy from* ··· command as shown in Chapter 4, Section 4.3.2.2. In the files, each line denotes a tuple. For every tuple, each attribute value is separated by a ";", while each instance of a set-triplet valued attribute is separated by a ",".

Attribute	Relation	Value Range	Time Range
ssno	Emp	1–500	–
name	Emp	E1–E500	–
address	Emp	A1–A500	–
salary	Emp	50K–120K	01/01/70–*now*
skills	Emp	S1–S10	01/01/70–*now*
dname	Emp	D1–D20	01/01/70–*now*
manager	Emp	M1–M20	01/01/70–*now*
dno	Dept	1–20	–
dname	Dept	D1–D20	–
budget	Dept	5000K–6000K	01/01/70–*now*
manager	Dept	M1–M20	01/01/70–*now*
pno	Proj	1–50	–
pname	Proj	P1–P50	–
budget	Proj	5000K–6000K	01/01/70–*now*
ssno	Assigned	1–500	–
pno	Assigned	1–50	–
type	Assigned	W1–W10	01/01/70–*now*
rating	Assigned	0–100	01/01/70–*now*

Figure 5.1: Attribute Ranges.

For the sake of convenience, two assumptions have been made while creating the historical data:

(1) All the tuples of a relation are homogeneous, i.e each set-triplet valued attribute of a tuple, has the same starting and ending times. Note that TDBMS supports non-homogeneous tuples. However, we do not expect that this restriction will have any adverse affects on the performance results.

(2) The instances of each set-triplet valued attribute are continuous, i.e we assume no null values in the tuples of relations.

For the set-triplet valued attributes, the time components of the temporal atoms are generated in an increasing manner from a uniform distribution. The major problem encountered in the generation of data was resolving the referential integrity between the *Emp* and *Dept* relations. While generating the department and manager values for each employee in the *Emp* relation, care should be taken that these values match with the corresponding ones in the *Dept* relation. For the snapshot database, this is not difficult to check. Once the *Dept* relation has been created, for each tuple of the *Emp* relation, when the department value is generated, the corresponding manager value is retrieved by going through the *Dept* relation and fetching the manager value for the respective department. Note that we consider manager of an employee and the manager of the department for which the employee works to be the same. However, resolving the referential integrity in the historical database is not that trivial. The algorithm used to generate data for the *Emp* relation is given in Figure 5.2. The algorithms used to generate the rest of the relations are similar and are not included here.

```
Procedure Rand-Emp(t,b)
begin                /* Rand-Emp */
   for the number of tuples $t$ do
      Randomly generate the $ssno$, $name$, and $address$ values
      for the number of instances $b$ do
         Randomly generate the $salary$ and $skills$ values
St1: for the number of instances $b$ do
         Randomly generate the time component and value of $dname$
         Let the start time be $t_0$
St2:     Search the $Dept$ file for the corresponding $dname$ value
         if found then
            Let the start time of the $manager$ attr. in $Dept$ be $t_0'$
            if $t_0 \geq t_0'$ then
               for each intersecting triplet do
                  Form a new triplet for the $manager$ attr. of the $Emp$ file
               go to St1
            else
               Randomly generate a new $dname$ value
               go to St2
end              /* Rand-Emp */
```

Figure 5.2: Algorithm for randomly generating data for the *Emp* relation

5.3 Queries

The list of example queries used for the performance evaluation is given below. Each query is executed against different database types. **S** and **S/N** represent snapshot and snapshot/nested databases respectively. $\mathbf{H_C}$, $\mathbf{H_H}$ and $\mathbf{H_R}$ stand for the historical database. $\mathbf{H_C}$ query types use the current data values, whereas $\mathbf{H_H}$, use the entire history to select qualifying values. $\mathbf{H_R}$ query types are used in the aggregation queries to select values valid for a certain period of time.

Q1.0	**Point Query**
S	What is the salary of employee 1?
S/N	What are the salary values of employee 1?
H_C	What is the current salary of employee 1?
H_H	What is the salary history of employee 1?
Q1.1	**Point Query**
	Same as **Q1.0**, except employee is 250.
Q1.2	**Point Query**
	Same as **Q1.0**, except employee is 500.
Q2	**Point Query**
S	What are the E#s of employees who make more than 60K?
S/N	What are the E#s of employees any value of whose salary is more than 60K?
H_C	What are the E#s of employees currently making more than 60K?
H_H	What are the E#s of employees who make more than 60K in their salary history?
Q3	**Range Query**
S	What are the E#s of employees who make between 50K and 70K?
S/N	What are the E#s of employees any value of whose salary is between 50K and 70K?
H_C	What are the E#s of employees currently making between 50K and 70K?
H_H	What are the E#s of employees who make between 50K and 70K in their salary history?
Q4	**Join between two Atomic attributes**
S	What is the budget of P# 7, and what are the E#s and types of work of the employees assigned to it?
S/N	What are the budget values of P# 7, and what are the E#s and types of work values of the employees assigned to it?
H_C	What is the current budget of P# 7, and are what the E#s and current types of work of the employees assigned to it?
H_H	What is the budget history of P# 7, and what are the E#s and histories of the types of work of the employees assigned to it?

Q5	**Join between two Atomic attributes**
S	What are the E#s of employees who worked in project name P#1?
H_H	What are the E#s of employees who worked in project name P#1?
S/N	What are the E#s of employees who worked in project name P#1?
Q6	**Join between two Atomic attributes**
S	What is the type of work done by employees whose salary is 60K?
S/N	What is the set of type of work done by employees any of whose salary is 60K?
H_C	What is the type of current done by employees whose current salary is 60K?
H_H	What is the history of type of work done by employees who made 60K at any time?
Q7	**Join between two Atomic attributes**
S	What is the type of work done by employees whose salary is $\geq$ 60K?
S/N	What is the set of type of work done by employees any of whose salary is $\geq$ 60K?
H_C	What is the type of current done by employees whose current salary is $\geq$ 60K?
H_H	What is the history of type of work done by employees who made $\geq$ 60K at any time?
Q8	**Join between two set-triplet attributes**
S	What are the D#s and P#s of the departments and projects which have the same budget?
S/N	What are the D#s and P#s of the departments and projects which have the same budget values?
H_C	What are the D#s and P#s of the departments and projects whose current budget is the same?
H_H	What are the D#s and P#s of the departments and projects which have the same budget values at the same time?

Q9 Join between two set-triplet attributes

S	What is the budget of the department for which E#123 is working?
S/N	What are the budget values of any department for which E#123 is working?
H_C	What is the current budget of the departments for which E#123 is currently working?
H_H	What is the budget history of the departments for which E#123 worked at any time?

Q10 Selection on a set-triplet attr., retrieval from another set-triplet attr.

S	What are the salaries of employees working in D#7?
S/N	What were the salary values of employees when they were working in D#7?
H_C	What are the current salaries of employees currently working in D#7?
H_H	What were the salaries of employees when they were working in D#7?

Q11 Aggregation (1)

S	What was the average salary of employees in each department?
S/N	What was the average salary of employees in each department?
H_C	What is the current average salary of employees currently working in each department?
H_H	What was the average salary of employees in each department?
H_R	What was the average salary of employees in each department for the years 1985 and 1986?

Q12 Aggregation (2)

S	What are the number of projects each employee has been assigned to?
S/N	What are the number of projects each employee has been assigned to?
H_H	What are the number of projects each employee has been assigned to?

Q13 Aggregation (3)

S	Which department has the maximum budget?
S/N	Which department has the maximum budget?
H_C	Which department has currently the maximum budget?
H_H	Which department has the maximum budget?
H_R	Which department had the maximum budget in 1985 and 1986?

Q14 Aggregation (4)

S	What was the average rating of each employee for the projects he was assigned to?
S/N	What was the average rating of each employee for the projects he was assigned to?
H_C	What was the current average rating of each employee for the projects he was assigned to?
H_H	What was the average rating of each employee for the projects he was assigned to?
H_R	What was the average rating of each employee for the projects he was assigned to in 1985 and 1986?

Q15 Aggregation (5)

S	What was the highest salary earned by each employee?
S/N	What was the highest salary earned by each employee?
H_H	What was the highest salary earned by each employee?

5.4 Database Parameters

The sample queries given in Section 5.3 are run under TDBMS against snapshot, snapshot/nested and historical databases. These databases have been generated in TDBMS. We perform our experiments by changing the relation size and number of triplets in set-triplet valued attributes as follows:

- Parameter set **A**
  The relations in this set have the following sizes:

  Emp – 500 tuples
  Dept – 20 tuples
  Proj – 50 tuples
  Assigned – 2000 tuples

  To see the effect of the amount of history on the performance of different databases, the sample queries are run against S, S/N, H_C and H_H databases with the above number of tuples. We consider 3 cases with varying amounts of historical data. Number of triplets in a set-triplet valued attribute indicates the volume of historical data.

A1. 10
A2. 5
A3. Some 5, some 10

In case **A1**, a set-triplet valued attribute contains 10 triplets whereas in case **A2**, it contains 5 triplets. Thus, the former represents a database which carries a longer history and the latter represents a shorter history. Number of triplets in a set-triplet valued attribute also gives an indication of how fast the values of an attribute change. For case **A3**, the aim is to see the performance variation which results in queries which involve a join between two set-triplet valued attributes, when one of the set-triplet attributes has 5 instances and the other 10 instances. The relations and the number of instances in the set-triplet attributes are given below.

Emp(ssno, name, address, *$salary(5), *$skills(10), *$dname(10), *$manager(10))
Dept(dno, dname, *$budget(5), *$manager(10))
Proj(pno, pname, *$budget(10))
Assigned(ssno, pno, *$type of work(5), *$rating(10))

Processing times of the sample queries for runs **A1**, **A2** and **A3** are given in Table 5.1 respectively.

- Parameter set **B**
  In this case, the same runs are repeated with a different database size where the number of tuples of *Emp* and *Assigned* have been reduced by half. Relations *Dept* and *Proj* remain the same as they are relatively small relations. This run gives an idea on how the performance of the different databases vary with the sizes of relations. The relations in this run have the following sizes:

Emp	– 250 tuples
Dept	– 20 tuples
Proj	– 50 tuples
Assigned	– 1000 tuples

The sample queries are run against **S**, **S/N**, $\mathbf{H_C}$ and $\mathbf{H_H}$ on the above number of tuples with the set-triplet valued attributes having the number of instances given in parameter set **A**. Processing times of the sample queries for runs **B1**, **B2** and **B3** are given in Table 5.2 respectively.

Q	S	S/N	H_C	H_H	H_R
1.0	0.38	0.65	2.45	0.92	–
1.1	0.46	0.63	2.43	0.85	–
1.2	0.33	0.68	2.43	0.82	–
2	0.60	19.55	2.57	78.20	–
3	0.41	19.03	2.53	82.25	–
4	3.61	2.42	3.28	3.68	–
5	3.54	2.23	–	2.52	–
6	3.14	21.02	9.77	81.20	–
7	3.48	25.98	10.30	98.96	–
8	1.07	0.87	1.52	0.88	–
9	1.43	18.85	3.90	83.65	–
10	0.48	1.20	3.52	78.60	–
11	0.65	182.43	3.70	?	?
12	1.17	1.23	–	1.57	–
13	0.32	1.38	0.57	1.45	1.60
14	1.63	32.63	7.35	123.30	133.71
15	0.71	20.37	–	83.00	–

A1

Q	S	S/N	H_C	H_H	H_R
1.0	0.27	0.53	1.64	0.60	–
1.1	0.33	0.45	1.64	0.60	–
1.2	0.37	0.53	1.58	0.62	–
2	0.38	6.20	1.78	18.93	–
3	0.37	6.12	1.40	19.68	–
4	2.60	2.07	2.58	2.33	–
5	2.70	1.93	–	2.17	–
6	2.33	8.02	6.32	21.88	–
7	2.30	10.52	7.03	30.07	–
8	0.80	0.82	1.45	0.90	–
9	1.08	6.57	2.58	22.23	–
10	0.32	5.83	2.15	19.33	–
11	0.47	30.67	2.25	101.12	107.48
12	0.70	1.03	–	1.03	–
13	0.25	0.55	0.55	0.80	0.97
14	1.02	12.00	4.63	26.79	39.63
15	0.53	6.17	–	19.95	–

A2

Q	S	S/N	H_C	H_H	H_R
1.0	0.28	0.72	1.90	0.87	–
1.1	0.33	0.58	1.77	0.82	–
1.2	0.37	0.60	1.82	0.82	–
2	0.45	9.40	2.05	37.13	–
3	0.37	9.37	1.93	37.57	–
4	2.68	2.22	2.85	2.73	–
5	2.57	2.10	–	2.32	–
6	2.27	11.43	7.27	39.70	–
7	2.38	14.18	7.50	48.90	–
8	0.73	0.78	1.60	0.95	–
9	1.07	16.17	3.47	66.68	–
10	0.32	16.23	2.95	61.83	–
11	0.43	85.06	3.05	?	?
12	0.78	1.10	–	1.40	–
13	0.28	0.62	0.60	0.80	0.88
14	1.00	23.77	7.12	79.18	90.88
15	0.53	9.93	–	46.06	–

A3

Table 5.1: Processing times (in seconds) for run **A**

Q	S	S/N	H_C	H_H	H_R
1.0	0.3	0.45	1.52	0.5	–
1.1	0.27	0.48	1.45	0.57	–
1.2	0.32	0.45	1.38	0.5	–
2	0.35	9.97	1.53	39.83	–
3	0.32	9.85	1.50	39.90	–
4	1.60	1.77	2.13	1.98	–
5	1.77	1.73	–	1.77	–
6	1.28	11.36	5.62	41.29	–
7	1.21	12.27	5.92	45.58	–
8	0.82	0.88	1.57	0.95	–
9	1.03	10.20	2.52	42.50	–
10	0.27	9.62	2.00	38.80	–
11	0.33	91.93	2.10	?	?
12	0.60	0.78	–	1.0	–
13	0.27	0.82	0.67	1.38	1.60
14	0.67	16.53	3.83	59.63	64.67
15	0.43	10.18	–	40.75	–

B1

Q	S	S/N	H_C	H_H	H_R
1.0	0.28	0.40	1.15	0.43	–
1.1	0.27	0.42	1.05	0.42	–
1.2	0.27	0.37	1.05	0.42	–
2	0.38	3.22	1.23	9.83	–
3	0.31	3.22	1.07	9.85	–
4	1.72	1.67	1.87	1.78	–
5	1.70	1.57	–	1.53	–
6	1.32	4.60	3.85	11.26	–
7	1.40	5.00	4.23	13.53	–
8	0.73	0.76	1.51	0.72	–
9	1.03	3.85	2.15	11.88	–
10	0.27	3.15	1.36	9.97	–
11	0.38	15.50	1.40	51.28	54.33
12	0.52	0.73	–	0.72	–
13	0.28	0.67	0.57	0.88	0.98
14	0.71	6.17	2.72	16.37	19.88
15	0.40	3.35	–	10.18	–

B2

Q	S	S/N	H_C	H_H	H_R
1.0	0.23	0.42	1.13	0.55	–
1.1	0.27	0.45	1.20	0.52	–
1.2	0.28	0.42	1.15	0.55	–
2	0.32	4.73	1.20	16.45	–
3	0.33	4.78	1.20	16.87	–
4	1.67	1.68	2.08	1.87	–
5	1.65	1.50	–	1.65	–
6	1.35	6.03	4.20	18.22	–
7	1.33	6.85	4.43	20.58	–
8	0.80	0.83	1.47	0.95	–
9	1.00	8.20	2.60	33.20	–
10	0.25	7.58	1.72	29.55	–
11	0.37	40.47	1.80	?	?
12	0.48	0.77	–	0.83	–
13	0.27	0.62	0.60	0.80	0.93
14	0.75	11.77	3.92	33.87	39.67
15	0.40	4.95	–	16.95	–

B3

Table 5.2: Processing times (in seconds) for run **B**

5.5 Experiments and Results

5.5.1 Comparison of Different Database Types

All experiments are based on runs **A** and **B** given in Section 5.4, and sample queries listed in Section 5.3

Experiment 1: Comparison of $\mathbf{S} - \mathbf{H_C}$

The graphs in Figure 5.3 and Figure 5.4 show how the performance degrades in moving from snapshot to historical databases for the sample queries. In moving from graph **SA1–HcA1** to graphs **SA2–HcA2** and **SA3–HcA3** of Figure 5.3, we see that decreasing the amount of history kept, decreases the performance degradation. The same is the case in Figure 5.4. The queries in Figure 5.4 have better processing times than those in Figure 5.3 since the *Emp* and *Assigned* relations have half the number of tuples. Furthermore, as a rough observation, the graphs in Figure 5.3 and Figure 5.4 show a consistent difference between the processing times of snapshot and current queries. Naturally, there are slight variations for the same query in different database types.

Experiment 2: Comparison of $\mathbf{H_C} - \mathbf{H_H}$

The aim of this experiment is to show how the performance varies in executing the sample queries on current values as compared to executing them on the entire history of the values.

As seen from the graphs in Figure 5.5 and Figure 5.6, the processing time of each query for $\mathbf{H_H}$ is considerably greater than that for $\mathbf{H_C}$. This is because for the $\mathbf{H_H}$ type queries, we first *unpack* set-triplet attributes of the concerned relations and then fetch the valid attribute values. In contrast, for the $\mathbf{H_C}$ type queries, we use the *Enum1* operation to select the current attribute value of the set-triplet attribute instead of using the *unpack* operation and then selecting the current value. As expected, reducing the size of the *Emp* and *Assigned* relations, in Figure 5.6, gives better performance results. For **Q9**, which involves a *join* between two set-triplet valued attributes, we see that for the $\mathbf{H_H}$ query, the performance improves when the amount of history in one of the set-triplet valued attribute is reduced. This is seen in moving from graph **HcA1–HhA1** to graph **HcA3–HhA3** of Figure 5.5 and Figure 5.6.

Experiment 3: Comparison of $\mathbf{S/N} - \mathbf{H_H}$

This experiment gives a measure of the overhead introduced by adding time to databases having relations which support set-valued attributes. As seen from Figure 5.7 and Figure 5.8, the processing time of each query for $\mathbf{H_H}$ is greater than that for **S/N**. This is due to the time involved in compressing the temporal attributes while storing them in the system, and uncompressing them when executing temporal algebra expressions. The reasoning for the processing time of **Q9** when moving from **S/NA1−HhA1** to graph **S/NA3−HhA3** of Figure 5.7 and Figure 5.8 in Experiment 2 holds here as well.

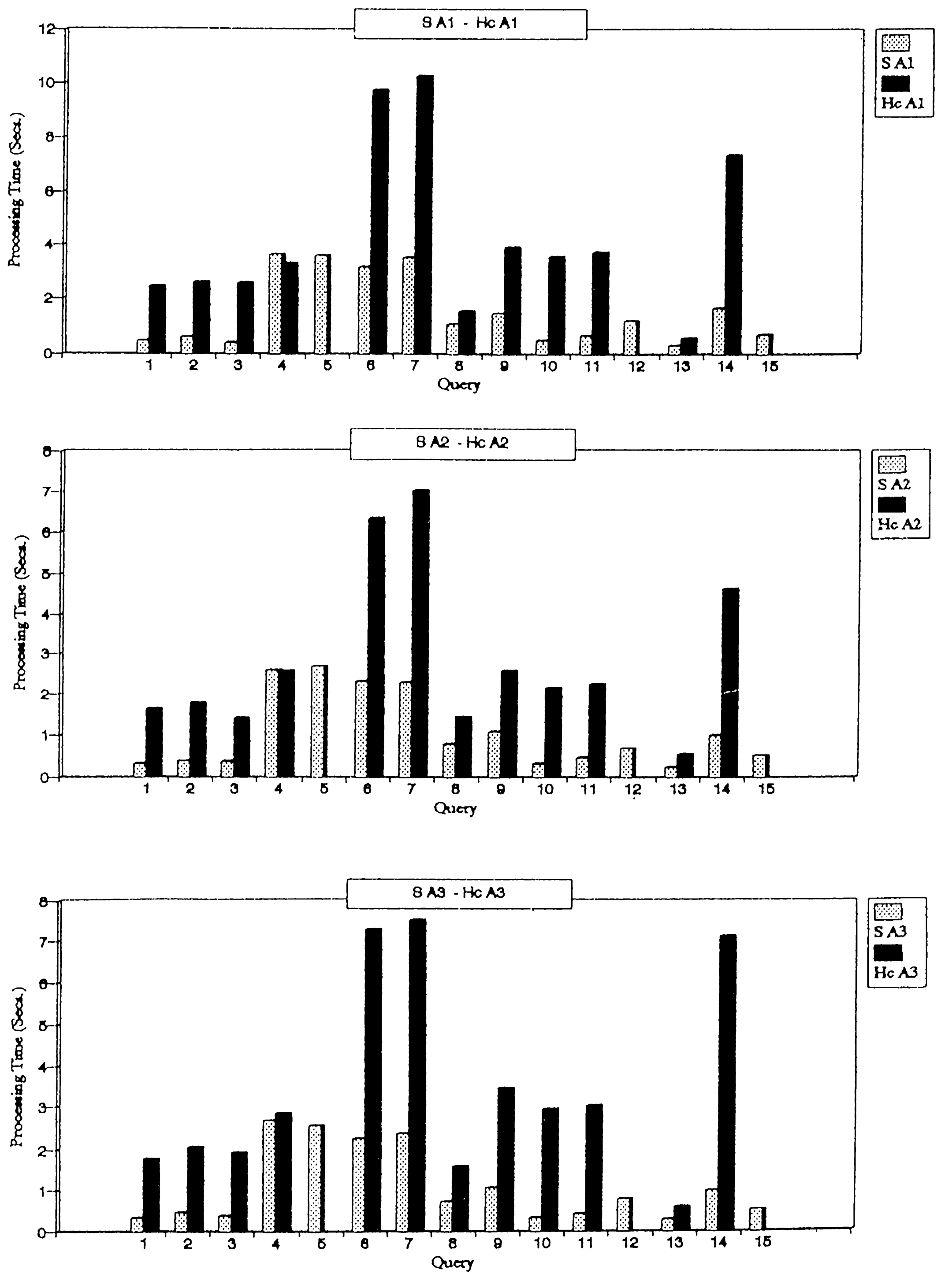
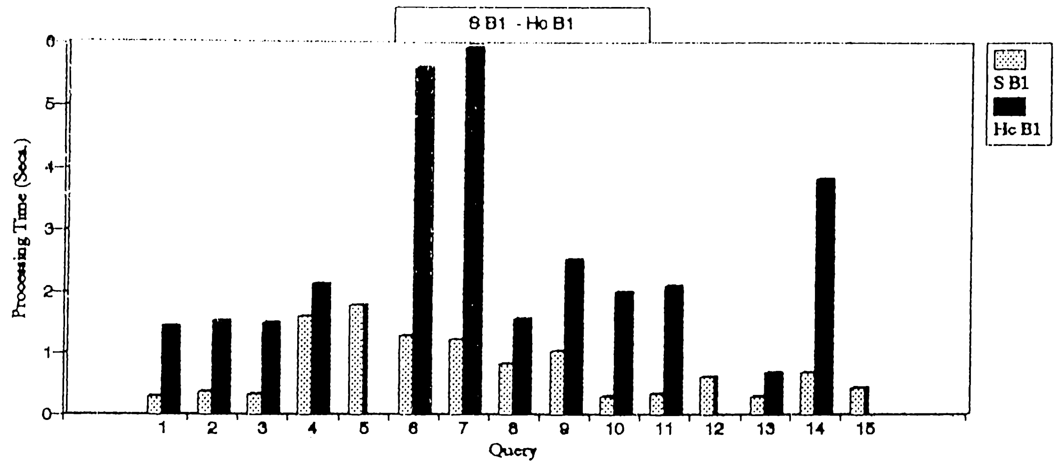
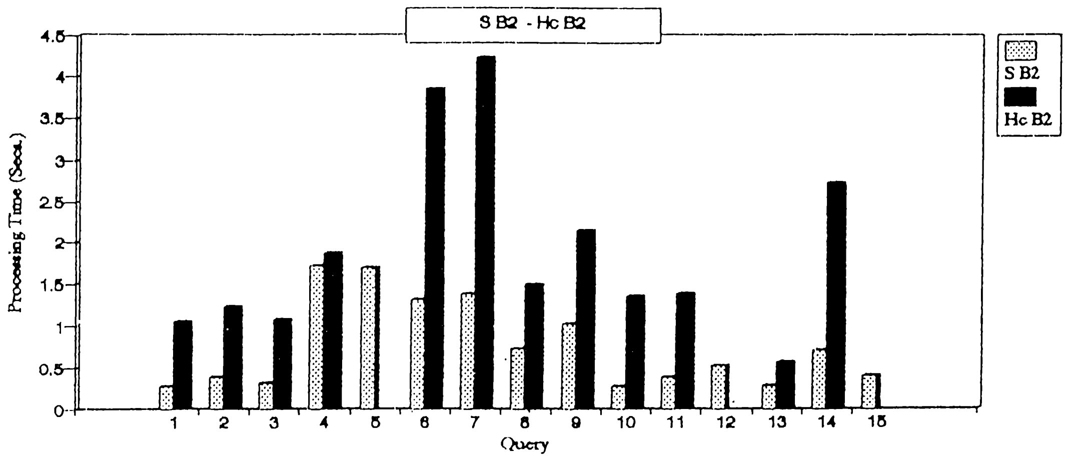


Figure 5.3: Experiment 1 — Run **A**





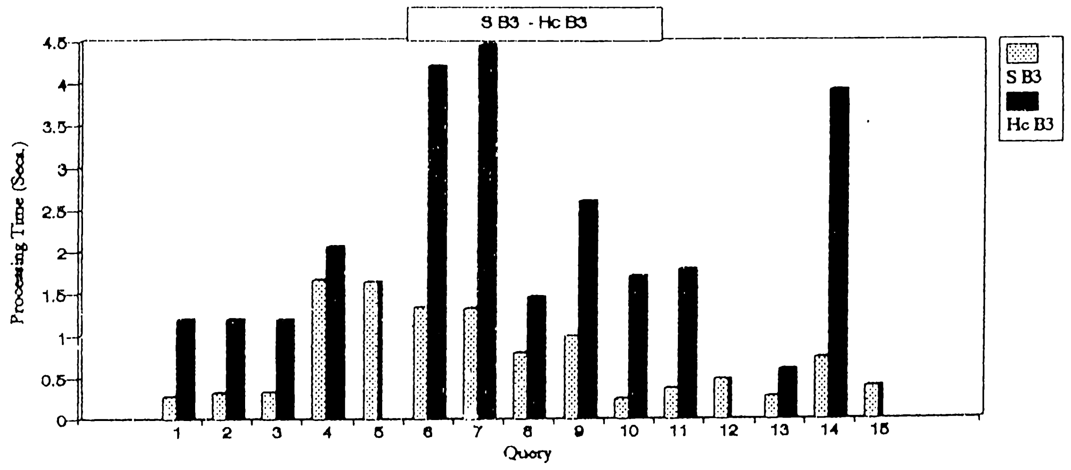


Figure 5.4: Experiment 1 – Run B

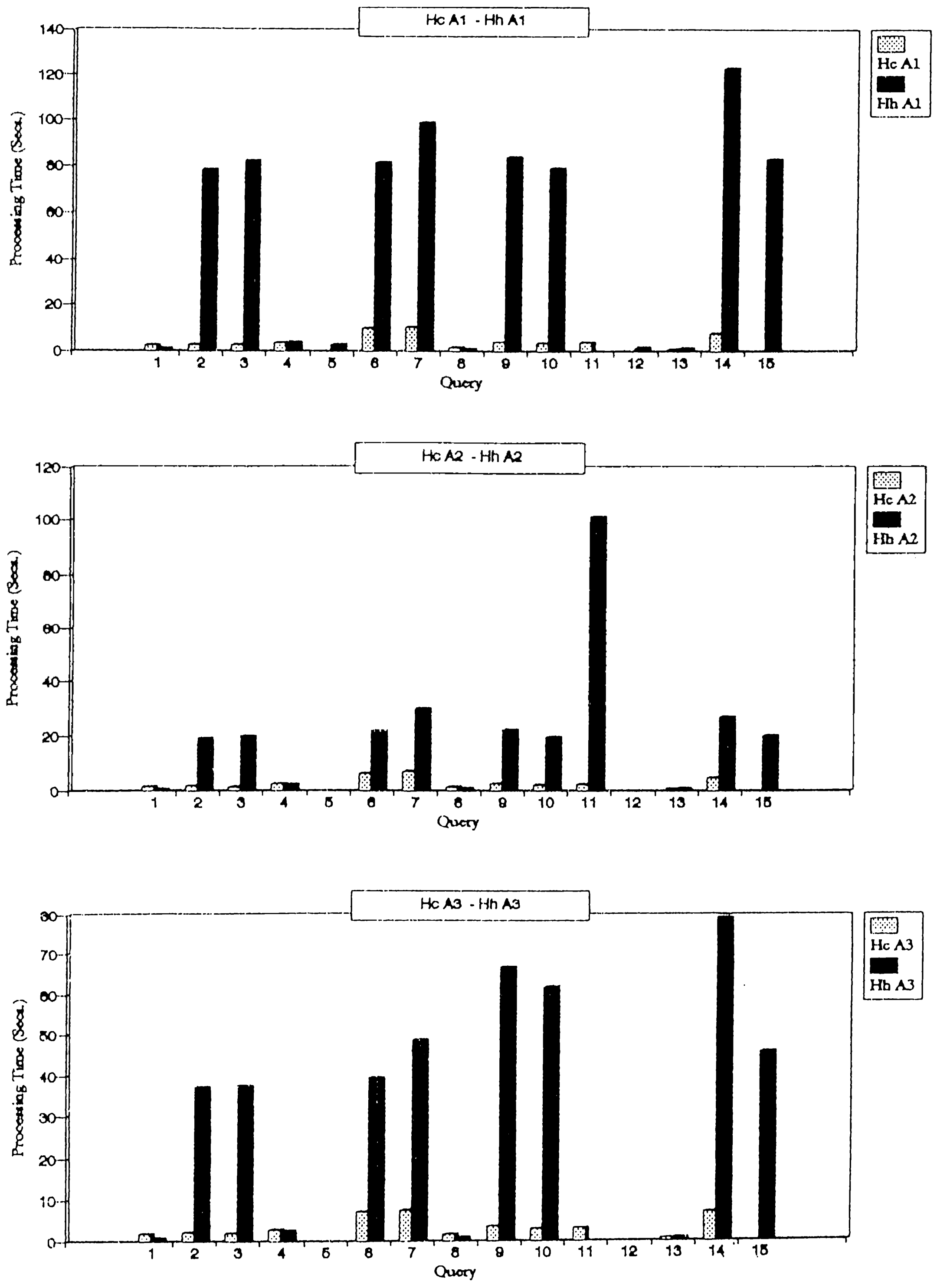


Figure 5.5: Experiment 2 -- Run **A**

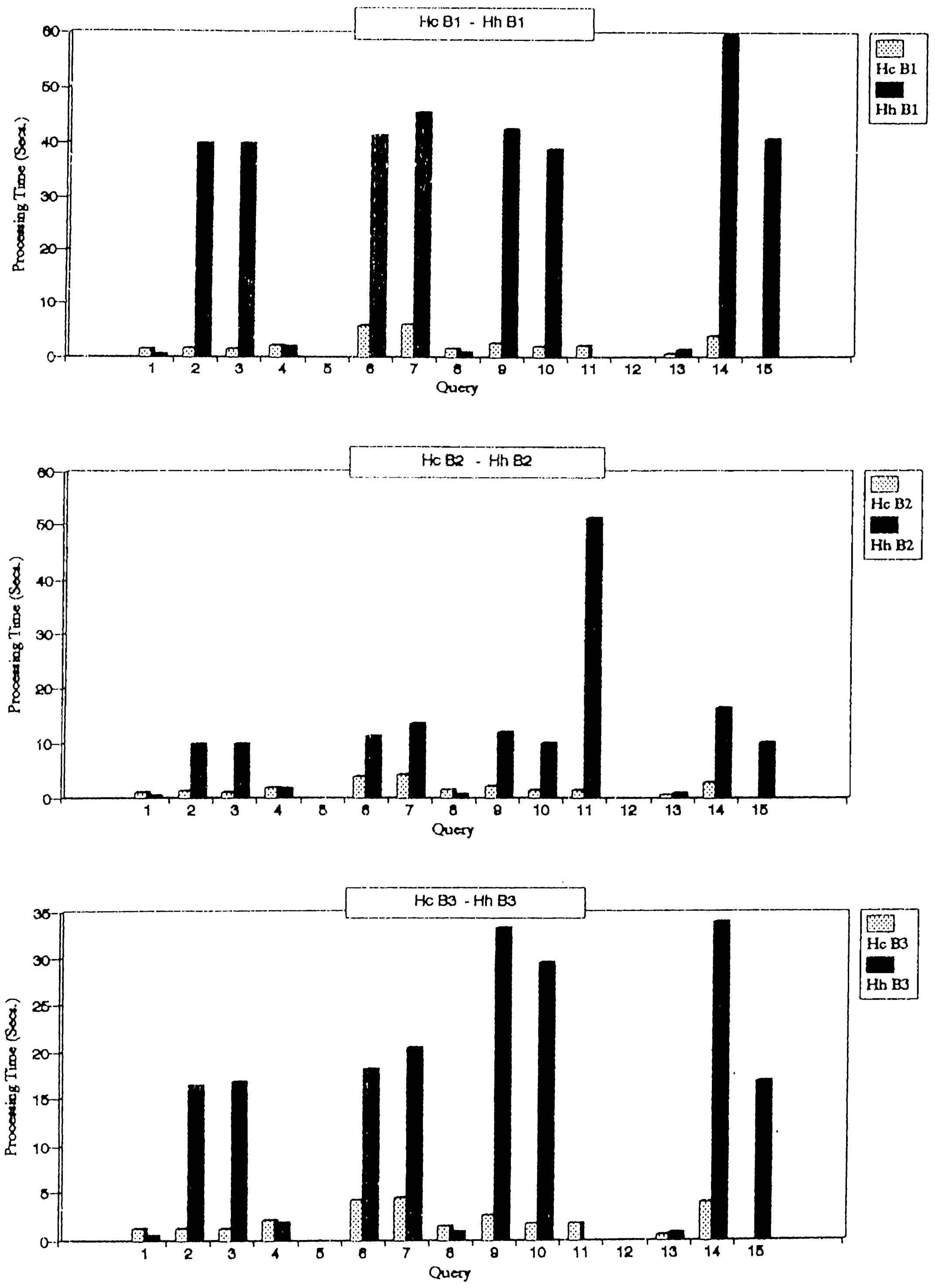


Figure 5.6: Experiment 2 – Run B

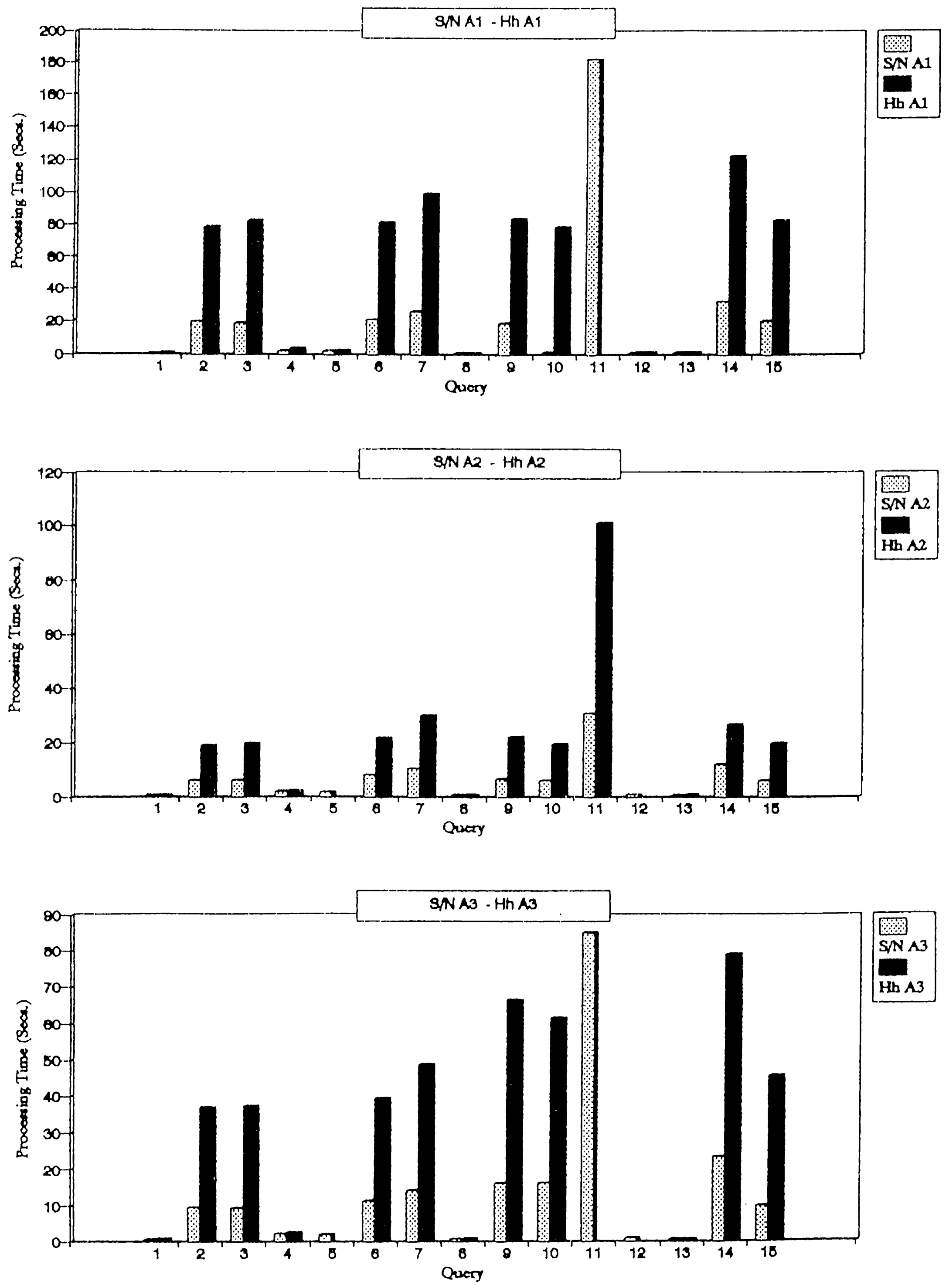


Figure 5.7: Experiment 3 — Run **A**

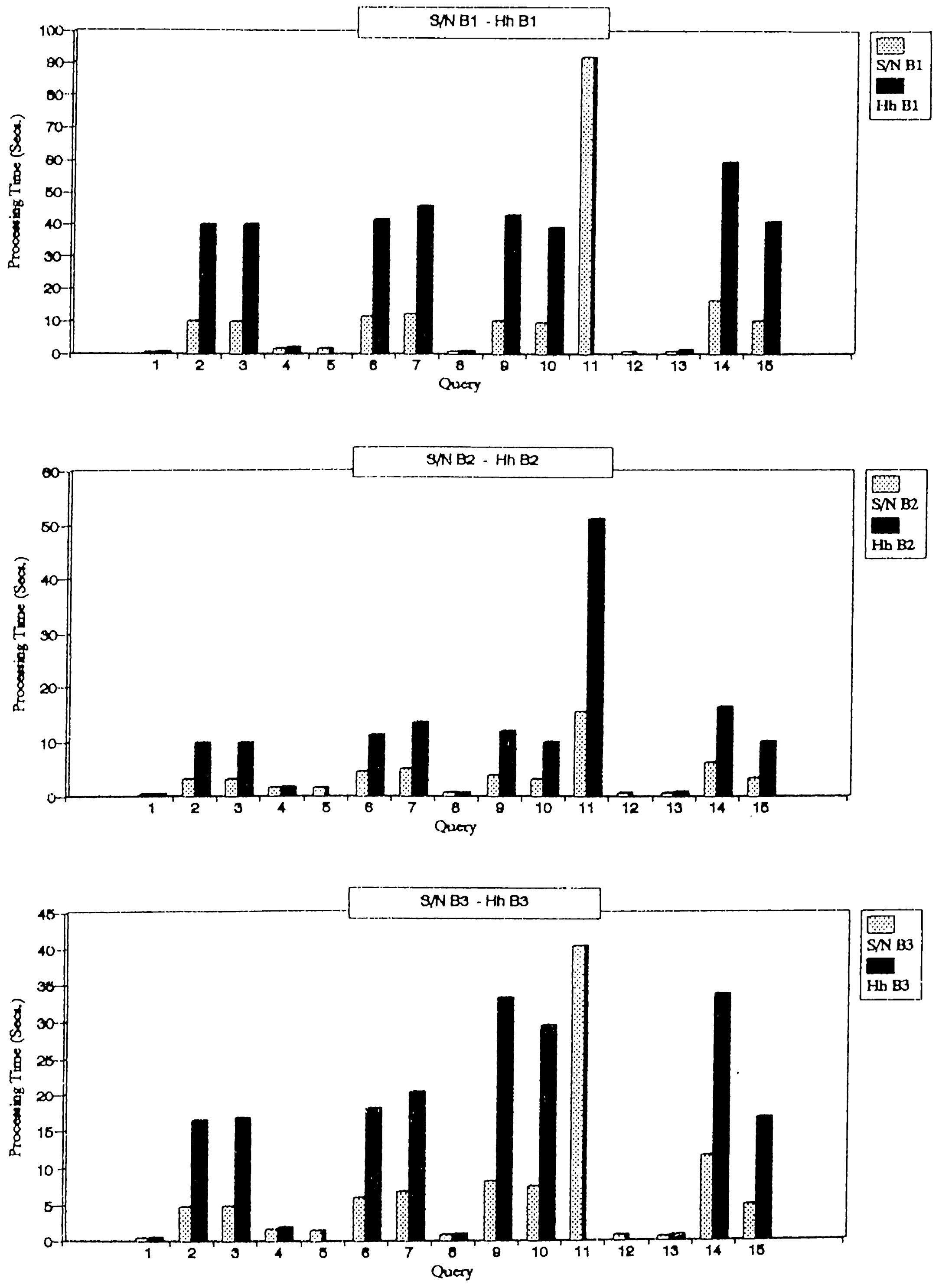


Figure 5.8: Experiment 3 – Run B

5.5.2 Comparison of Databases on Varying the Size of Histories

In this section, we compare the performance of the sample queries on the individual database types **S/N**, $\mathbf{H_C}$ and $\mathbf{H_H}$ by varying the amount of history.

Experiment 4: S/N

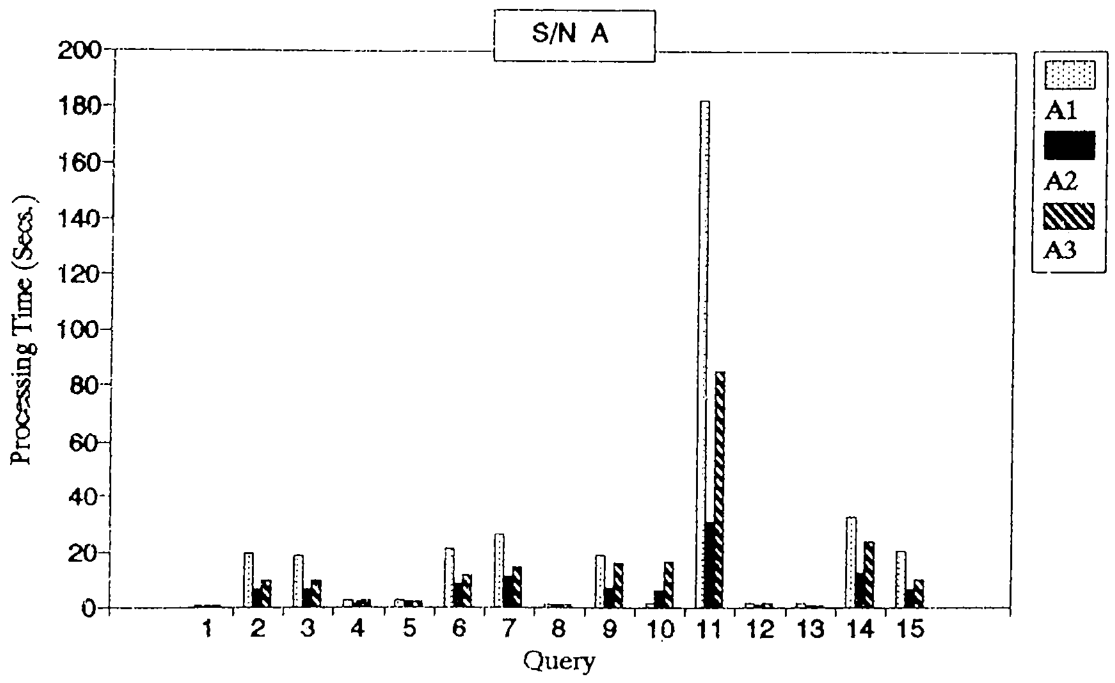
The graphs in Figure 5.9 show that the performance varies considerably for all queries when the size of the snapshot-nested database is changed. As expected, reducing the size of the database improves the processing time of the queries as seen in graphs **S/NA** and **S/NB**. In moving from graph **S/NA** to **S/NB**, we see that relations with a smaller size, as expected have better performance times.

Experiment 5: $\mathbf{H_C}$

Although reducing the amount of history improves the processing time of the sample queries, as seen in graphs **HcA** and **HcB** of Figure 5.10, the variation in the performance is not as large as that in **S/N** and $\mathbf{H_H}$. This is due to using the *Enum1* operation to select the current values from the set-triplet valued attribute as opposed to using the ***unpack*** operation and then selecting the current values from the set-triplet valued attribute. Similar to Experiment 4, relations with a smaller size have better performance times.

Experiment 6: $\mathbf{H_H}$

Similar to Experiment 4, processing time of the queries reduces considerably when the amount of history decreases as seen in graphs **HhA** and **HhB** of Figure 5.11. **Q11** could not be executed for runs **A** and **B**, since the *unpacking* of the *Emp* resulted in very large relations. Similar to Experiment 4 and 5, reducing the size of the relations gives better performance times. Furthermore, the variation in performance of queries in this case is much higher than that of Experiments 4 and 5. This is partly due to the size of intermediate relations created during query processing.



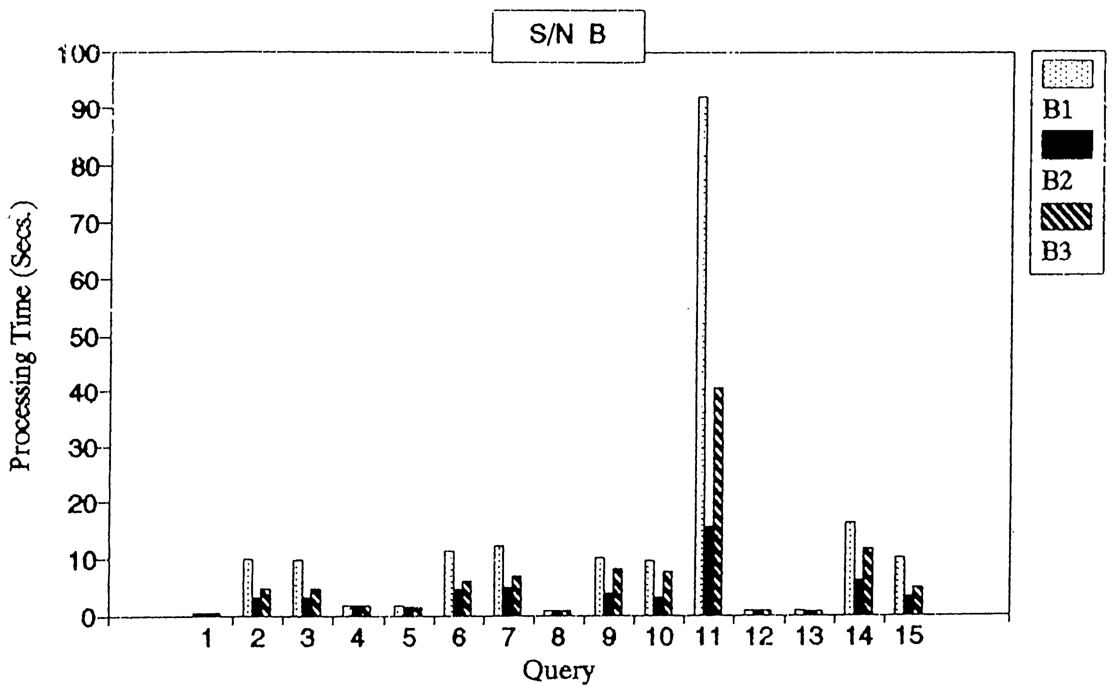
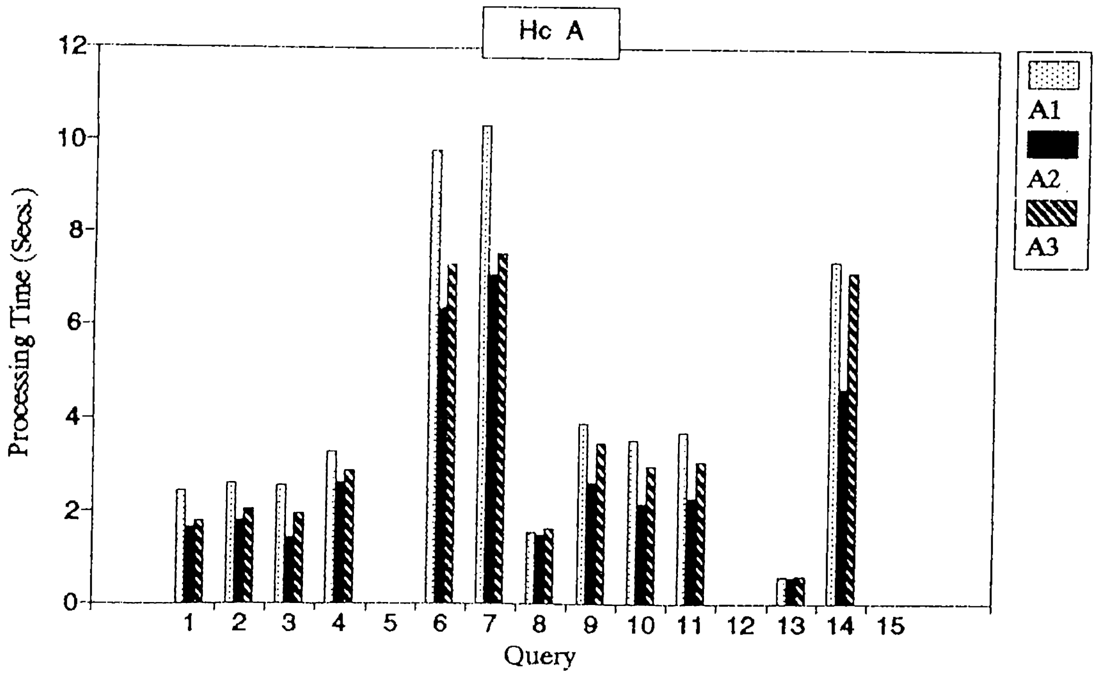


Figure 5.9: Experiment 4: S/N – Runs **A** and **B**



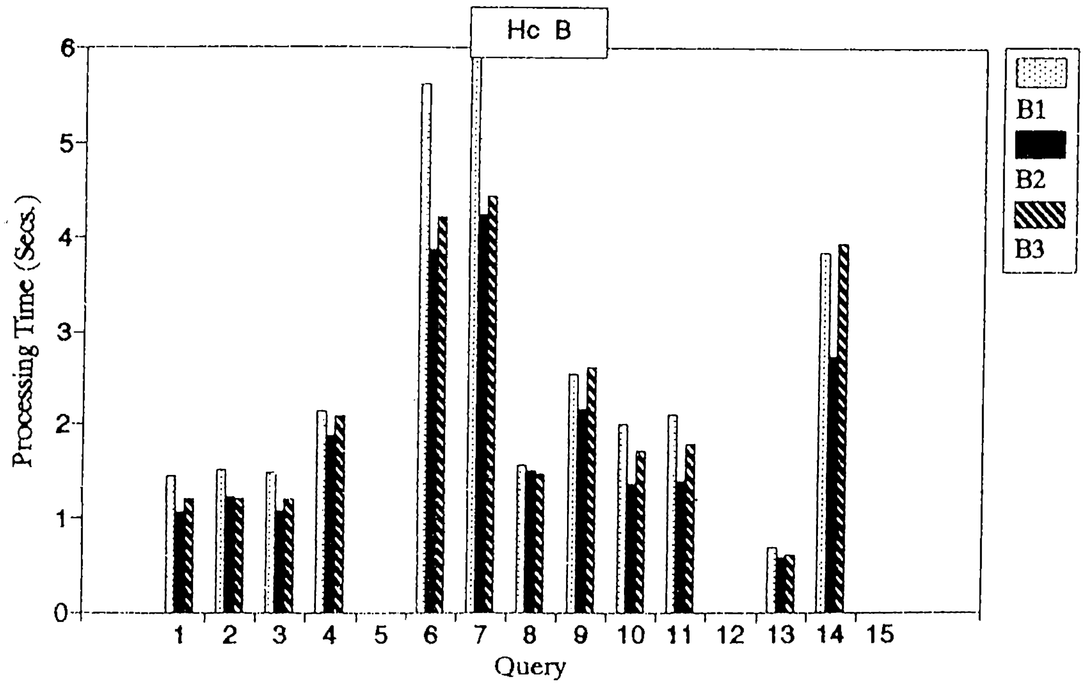
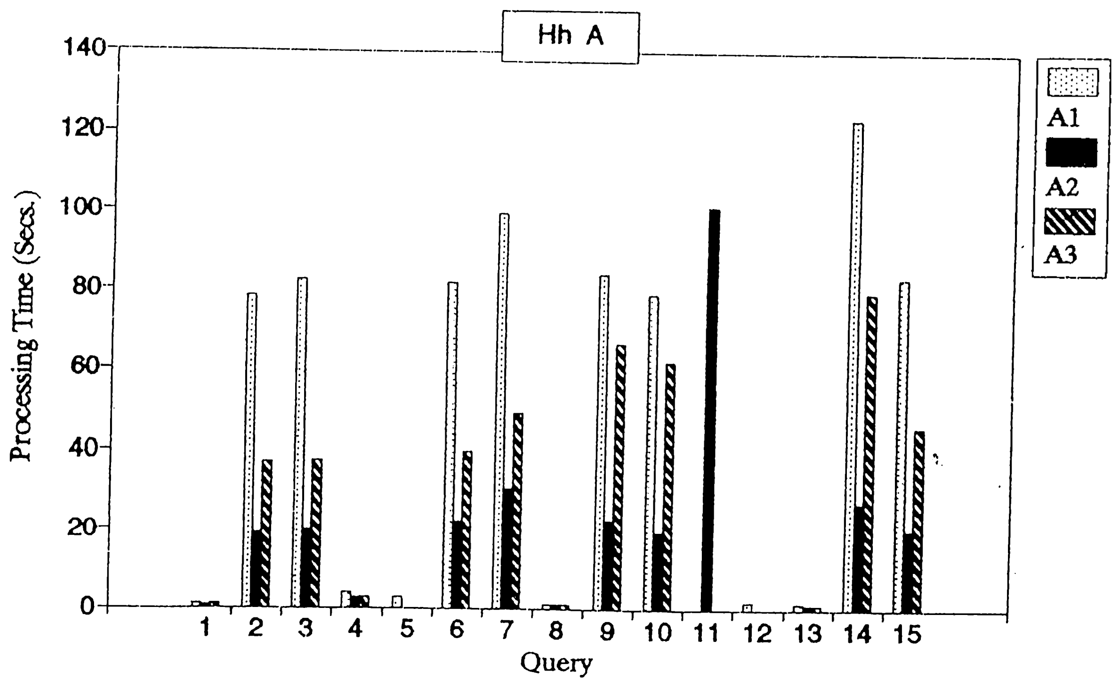


Figure 5.10: Experiment 5: H_C – Runs **A** and **B**



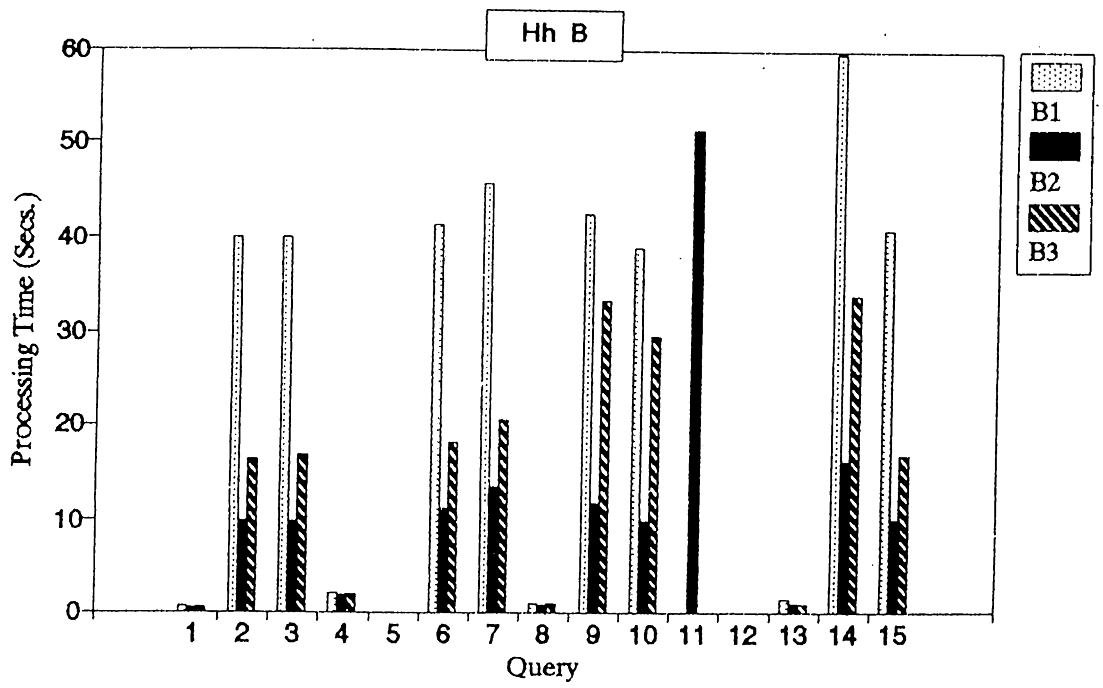


Figure 5.11: Experiment 6: $\mathbf{H_H}$ – Runs **A** and **B**

Chapter 6

Conclusion

In this work, we describe the implementation of the temporal relational database management system (TDBMS) on top of an existing relational database management system (ERAM) [9]. ERAM provides a natural environment for the implementation of the methodology introduced for incorporating time dimension into the relational model as proposed in [31].

The implemented system supports a time granularity of days and allows different types of attributes to coexist: atomic values, triplets and their sets, and handles historical data in a convenient way. TDBMS consists of revised versions of the basic set of operations of the extended relational algebra, and new operations which convert one attribute type to another and do selection over the time dimension.

Moreover, a statistical interface has been added to TDBMS. This interface includes aggregate functions and transformations of data into tabular forms suitable for advanced statistical analysis. The *enumeration* operation, which derives a table of uniform data for a set of specified time points or intervals, from a three dimensional historical relation is also included in the interface.

A performance evaluation of the system has been carried out on different types of databases. Sample queries have been run against snapshot databases, historical databases and databases that allow set-valued attributes. Performance degradation in moving from snapshot to historical databases has occurred. In addition, the overhead introduced by adding time to databases which allow set-valued attributes was also noticed. The absence of a time index and appropriate access paths account for the performance variation.

This prototype is the first implementation where attribute time stamping is used. It will be useful in testing the theory developed for temporal databases. It will also form the foundation for future implementation of temporal query languages, i.e HQUEL [33], TBE [34], temporal query optimization, performance evaluation of alternative models, etc.

Further research should be carried out for the addition of a time index to improve processing of certain class of temporal queries. Other database functions such as concurrency control, crash recovery, etc., are also among the topics that could be incorporated to the prototype implemented. Problems created by historical databases in handling these functions also deserve further investigation.

Bibliography

[1] Ariav, G., "A Temporally Oriented Data Model", ACM TODS, Vol.11, No.4, 1986.

[2] Ahn, I., Snodgrass, R., "Performance Evaluation of a Temporal Database Management System", Proc. of ACM SIGMOD Conf., May 1986, pp 96-107.

[3] Ahn, I., Snodgrass, R., "Performance Analysis of Temporal Queries", Information Sciences, Vol.49, 1989, pp 103-143.

[4] Bolour, A., Anderson, T.L., Dekeyser, L.J., Wong, H.K.T., "The Role of Time in Information Processing: A Survey", ACM SIGMOD Record. Vol.12 No.3 (1982) pp 28-48.

[5] Codd, E.F., "A Relational Model of Data for Large Shared Data Banks", Comm. of ACM. Vol.13 No.6 (1970) pp 377-387.

[6] Clifford J., Tansel, A.U., "On a Historical Relational Algebra: Two Views", Proc. of ACM SIGMOD Conf., 1985, pp 247-265.

[7] Elmasri, R., Wuu, G., "A Temporal Model and Query Language for ER Databases", Proc. of the Sixth IEEE International Conf. on Data Engineering, Feb. 1990, pp 76-83.

[8] Fisher, P., Thomas, S., "Operators for Non-First Normal Form Relations", Proc. of 7th Int. Computer Software Applications Conf., 1983.

[9] Hou, Wen-chi., "The Implementation of the Extended Relational Database Management System", MS thesis, Case Western Reserve University, 1985.

[10] Gadia, S.K., "A Homogeneous Relational Model and Query Languages for Temporal Databases", ACM TODS, Vol.13, No.4, 1988.

[11] Ginsburg, S., Tanaka, K., "Computational Tuple Sequences and Object Histories", Proc. of VLDB, 1984.

[12] Gadia, S.K., Yeung, C.S., "Inadequacy of Interval Timestamps in Temporal Databases", Information Sciences, Vol.54, 1991, pp 1-22.

[13] Jaeschke, G., Schek, H. J., "Remarks on the Algebra of Non-First Normal Form Relations", Proc. ACM Symposium on Principles of Database Systems, 1982.

[14] Khoshofian, S.N., Copeland, G., "Object Identity", Proc. of OOPSLA 86, ACM, New York, 1986.

[15] Klug, A. "Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions", Journal of ACM, July, 1982.

[16] Klopprogge, M.R., Lockemann, P.C., "Modelling Information Preserving Databases: Consequences of the Concept of Time", Proc. of VLDB, 1983.

[17] Lorentzos, N.A., Johnson, R.G., "Extending Relational Algebra to Manipulate Temporal Data", Information Systems, Vol.15, No.3, 1988.

[18] Mc Kenzie, E., Snodgrass, R., "Supporting Valid Time: A Historical Algebra", Technical Report TR87-008, Computer Science Department, UNC at Chapel Hill, 1987.

[19] Mc Kenzie, L.E. Jr., Snodgrass, R.T., "Evaluation of Relational Algebras Incorporating the Time Dimension in Databases", ACM Computing Surveys, Vol.23, No.4, Dec. 1991.

[20] Navathe, S.B., Ahmed, R., "TSQL - A Language Interface for History Databases", Conf. on Temporal Aspects of Information Systems", 1987.

[21] Ozsoyoglu, Z.M., Ozsoyoglu, G., "An Extension of Relational Algebra for Summary Tables", Proc. Second LBL Workshop on Statistical Database Management, 1983.

[22] Ozsoyoglu, G., Ozsoyoglu, Z.M., Matos, V., "Extending Relational Algebra and Relational Calculus with Set-Valued Attributes and Aggregate Functions", ACM Trans. Databababase Syst., Vol.12, No.4, Dec. 1987, pp 566-592.

[23] Sarda, N., "Extensions to SQL for Historical Databases", IEEE Trans. on Knowledge and Data Engineering, Vol.2, No.2, June 1990, pp 220-230.

[24] Shoshani, A., Kawagoe, K., "Temporal Data Management", VLDB 12, Kyoto, Japan, 1986.

[25] Segev, A., Shoshani, A., "Modelling Temporal Semantics", Temporal Aspects of Information Systems Conf., 1987.

[26] Snodgrass, R., "The Temporal Query Language, TQuel", ACM Trans. Database Syst., Vol.12, No.2 pp.247-298, June 1987.

[27] Snodgrass, R. (Ed.), "Research Concerning Time in Databases: Project Summaries", ACM SIGMOD Record, Vol.15, No.4, 1986.

[28] Snodgrass, R. (Ed.), IEEE Data Engineering, Vol.11, No.4, 1988.

[29] Snodgrass, R., "Temporal Databases: Status and Research Directions", ACM SIGMOD Record, Vol.19, No.4, Dec. 1990, pp 83-89..

[30] Soo, M.D., "Bibliography on Temporal Databases", ACM SIGMOD Record, Vol.20, No.1, March 1991, pp 14-23.

[31] Tansel, A.U., "Adding Time Dimension to Relational Model and Extending Relational Algebra", Information Systems Vol.13 No.4 (1986) pp 343-355.

[32] Tansel, A.U., "A Statistical Interface for Historical Relational Databases", Proc. of the Third IEEE International Conf. on Data Engineering. 1987, pp 538-546.

[33] Tansel, A.U., Arkun, M.E., "HQUEL: A historical query language", in Proc. 3^{rd} Int. Workshop on Statistical Sci. Database Management, Luxembourg, 1986.

[34] Tansel, A.U., Arkun, M.E., Ozsoyoglu,G., "Time-by-Example Query Language for Historical Databases", IEEE Trans. on Software Eng. Vol.15 No.4 April 1989.

[35] Tansel, A.U., Garnett, L., "Nested Historical Relations", Proc. of ACM SIGMOD Conf. 1989.

Appendix A

COMMAND SYNTAX

Some conventions used for the syntax are as follows:

(a) Bold-faced words/symbols are reserved keywords.

(b) Words which are in regular font are generic types which must be supplied by the user.

(c) Square brackets, [], indicate that the enclosed item is optional.

(d) Curly brackets, {}, indicate a repetition of the enclosed item.

(e) A slash, /, indicates an alternation.

Either blanks or commas can be used as delimiters for items in the input command stream.

- <u>CREATEDB</u> : **createdb** dbname
- <u>DELETEDB</u> : **deletedb** dbname
- <u>TDBMS</u> : **tdbms** dbname
- <u>CREATE</u> : **create** rel-name (atr-name := format {atr-name := format})
  [**Key** = atr-no {atr-no}]
- <u>DESTROY</u> : **destroy** rel-name {rel-name}
- <u>COPY</u> : **copy** rel-name ({atr-name}) **from/into** file-name
- <u>APPEND</u> : **append** rel-name (atr-name := value {atr-name := value})
- <u>DELETE</u> : **delete** rel-name **where** cond-exp

- <u>UPDATE</u> : **update** rel-name (atr-name := value {atr-name := value})
  where cond-exp
- <u>PRINT</u> : **print** [-l] (rel-name/alg-exp)
- <u>SCHEME</u> : **scheme** [-l] {rel-name}
- <u>RENAME</u> : **rename** rel-name ({atr-name := new-atr-name})
- <u>UNION</u> : (rel-name/alg-exp) **union** (rel-name/alg-exp)
- <u>DIFF</u> : (rel-name/alg-exp) **diff** (rel-name/alg-exp)
- <u>CPROD</u> : (rel-name/alg-exp) **cprod** (rel-name/alg-exp)
- <u>PROJECT</u> : **project** (rel-name/alg-exp) **on** atr-name {atr-name}
- <u>SELECT</u> : **select** {atr-name} **from** (rel-name/alg-exp) **where** cond-exp
- <u>NJOIN</u> : (rel-name/alg-exp) **njoin** (rel-name/alg-exp)
- <u>UNPACK</u> : **unpack** (rel-name/alg-exp) **on** atr-name
- <u>PACK</u> : **pack** (rel-name/alg-exp) **on** atr-name
- <u>TDEC</u> : **tdec** rel-name **on** atr-name
- <u>TFORM</u> : **tform** rel-name **on** atr-name atr-name atr-name
- <u>SLICE</u> : **slice** rel-name atr-name **by** atr-name
- <u>USLICE</u> : **uslice** rel-name atr-name **by** atr-name
- <u>DSLICE</u> : **dslice** rel-name atr-name **by** atr-name
- <u>DROPTIME</u> : **droptime** rel-name **on** atr-name
- <u>AGGREGATE FORMATION</u> : (rel-name/alg-exp) < {X} aggrf (atr-name) >
- <u>ENUMERATION 1</u> : **enum1** (rel-name/alg-exp) <{X_1} {X_2} > {T}
- <u>ENUMERATION 2</u> : **enum2** (rel-name/alg-exp) < {X} {aggrf(atr-name)} > {T}

Appendix B

USER MANUAL FOR THE TEMPORAL COMMANDS

This is the user manual for the temporal commands in TDBMS. Details for the rest of the commands can be found in [9]. Each command in the manual has the following format:

NAME section: Gives the command name.

SYNTAX section: We use the syntax conventions described in Appendix A.

DESCRIPTION section: This section gives a detailed description of the command. Requirements and the use of the command are outlined.

EXAMPLE section: This section gives one or more examples to illustrate the command and its uses. The examples are based on the relations given in Chapter 5, Section 5.2.1.

The commands, *createdb*, *deletedb*, *listdb* and *tdbms* are executed from Unix. The other commands are executed from inside the TDBMS system. Before issuing any commands, the user must be entered into the *users* file by the superuser. This will give the user authorization to access the system and create databases.

To start using the system type *tdbms* followed by a database name. The system will prompt a @ waiting for input. After typing a command, press *return* to execute the command. An appropriate error message is printed in case of a syntax or semantic error. The *help* command will print the related manual section of a command. To exit from the system, all you have to do is type *q*.

An alternative way to supply commands to the system is to provide an input text file which contains a series of commands. This can be done be *tdbms dbname* < *inputfile* >

outputfile. The command are read from *inputfile* and the output directed to *outputfile*.

Any command can be preceded by a *"rel-name :="* in order to store the output to a permenant relation. Otherwise, the default output device is the console. Generic names like relation names, attribute names cannot exceed 8 characters. Extra characters are truncated silently.

Finally, an output relation can be printed on the terminal, line printer or stored as a permanent relation.

The following is a description of all the temporal commands in the system:

- **NAME** : **AGGREGATE FORMATION**
  SYNTAX : (rel-name/alg-exp) < $\{X\}$ aggrf (atr-name) >
  DESCRIPTION :
  This command applies an aggregate function to tuples of each partition.The input relation can be either an existing relation, *rel-name* or the output of another extended relational algebra expression, *alg-exp*. *Aggrf* represents one of the aggregate functions - sum, avg, max, min, count and median. In addition, the functions first and last are included for temporal attributes. X is a set of attribute names. *Atr-name* is the name of an atomic attribute on which the *aggrf* is applied. The aggregate formation operator first partitions tuples of the relation *rel-name* such that tuples having the same X component are in the same partition. Then the aggregate function *aggrf* is applied to the component *atr-name* of tuples in each partition. The X-valued and associated value produced by the aggregate function are the output for each partition.

  EXAMPLE :

  /* Find the total salary of employees in each department and display it on the terminal */

  (tdec (unpack (unpack emp on salary) on dname) on dname)<{dname} sum(sal)>

  /* Which department has the maximum budget? */

  (unpack dept on budget)<{dno} max(budget)>

- **NAME** : **DROPTIME**
  SYNTAX : **droptime** rel-name **on** atr-name
  DESCRIPTION :
  This operation gets rid of the time of an attribute. The input relation can be either an existing relation *rel-name* or the output of another extended relational algebra expression *alg-exp*. *Atr-name* is the name of an attribute in the input relation of which the time is to be dropped. *Atr-name* should be a triplet-valued or set-triplet valued attribute. The former is converted to a normal atomic attribute while the

latter is converted to a set-valued attribute.

EXAMPLE

/* Droptime emp relation on attribute salary */

droptime emp on salary

- **NAME** : **DSLICE**
  SYNTAX : **dslice** rel-name atr-name **by** atr-name
  DESCRIPTION :
  This operation forms the difference slice of the time component of an attribute with respect to another. The input relation can be either an existing relation *rel-name* or the output of another extended relational algebra expression *alg-exp*. *Atr-names* should both be triplet-valued. The difference of the time components of the two attributes is assigned to the time component of the first attribute. The data types of the attributes remain unchanged. The attribute type of the first attribute is changed to *set-triplet* while that of the second attribute remains unchanged.

  EXAMPLE

  /* Difference slice salary attribute by dname attribute in emp relation */

  dslice (unpack(unpack emp on salary) on dname) salary by dname

- **NAME** : **ENUMERATION 1**
  SYNTAX : **enum1** (rel-name/alg-exp) <{X_1} {X_2} > {T}
  DESCRIPTION :
  This version of the enumeration operation selects designated attribute values at specified time points. The input relation can be either an existing relation *rel-name* or the output of another extended relational algebra expression *alg-exp*. X_1 is a set of attribute names to be part of output. X_2 is the set of temporal attribute names to which *enum1* is applied. The *enum1* operator, for each tuple of the relation *rel-name,* checks whether each time point in T intersects with the triplets of the temporal attributes in X_2. The attributes in X_1, the values of each attribute in X_2 that qualifies in the intersection and the time points in T form the result.

  EXAMPLE

  /* Find the salary values of employees on Jan 01, 1992 and display it on the terminal */

  enum1 emp<{ename} {salary}>{"010192"}

  /* Find the department names of each employee on Jan 01, 1991 and Jan 01, 1992 */

  enum1 emp<{ename} {dname}>{"010191" "010192"}

- **NAME** : **ENUMERATION 2**
  SYNTAX : **enum2** (rel-name/alg-exp) <{X} {aggrf(atr-name)} > {T}
  DESCRIPTION :
  This version of the enumeration operation applies aggregate functions to designated attribute values at specified time intervals. The input relation can be either an existing relation *rel-name* or the output of another extended relational algebra expression *alg-exp*. X is a set of attribute names to be part of output. $Agrf_i$, $1 \leq i \leq k$ is an aggregate function which is applied to $atr - name_i$ which is a temporal attribute name. The aggregate functions available for *enum2* are - sum, psum, avg, wavg, max, pmax, min, pmin, count, median, first, last and len. The *enum2* operator, for each tuple of the relation *rel-name*, checks whether each time interval in T intersects with each of the specified temporal attributes. The attributes in X, the values of the aggregated set of qualifying attributes and the time intervals of T are the output.

  EXAMPLE

  /* Find the sum of salary values of each employee in 1992 and display it on the terminal */

  enum2 emp<{ssno} sum(salary)>{"010192010193"}

  /* Find the maximum and minimum salary values of each employee in 1991 and 1992 */

  enum2 emp<{ename} max(salary) min(salary)>{"010191010192" "010192010193"}

- **NAME** : **PACK**
  SYNTAX : **pack** (rel-name/alg-exp) **on** atr-name
  DESCRIPTION :
  This operation packs a relation on an attribute. *Rel-name* is the name of an existing relation. *Alg-exp* represents the output relation of an extended relational algebra expression. Let C be the set of all attributes in input relation except *atr-name*. The *pack* operator maps set of tuples in input relation, whose components for attributes in C are the same into a single tuple. The C component of the packed tuple is the same as C component of those tuples that are packed. The value of *atr-name* in resulting tuple is the union of *atr-name* values of those packed tuples. The attribute *atr-name* in resulting relation is a nested attribute if the packing attribute was atomic, else it is set-triplet if the packed attribute was a triplet attribute.
  EXAMPLE :

  /* Display the different projects each employee is assigned to */

  pack (project assigned on ssno pno) on pno

- **NAME** : **SLICE**

SYNTAX : **slice** rel-name atr-name **by** atr-name
DESCRIPTION :
Slice operation slices the time component of an attribute with respect to another. The input relation can be either an existing relation *rel-name* or the output of another extended relational algebra expression *alg-exp*. *Atr-names* should both be triplet-valued. The intersection of the time components of the two attributes is assigned to the time component of the first attribute. If no intersection is found, the tuple is discarded from the result. The data and attribute types of the attributes remain unchanged.

EXAMPLE :

/* Slice salary attribute by dname attribute in emp relation */

slice (unpack(unpack emp on salary) on dname) salary by dept

- **NAME** : **TDEC**
  SYNTAX : **tdec** rel-name **on** atr-name
  DESCRIPTION :
  Triplet decomposition breaks a temporal attribute into its components. The input relation can be either an existing relation *rel-name* or the output of another extended relational algebra expression *alg-exp*. *Atr-name* is the name of an attribute in the input relation on which the triplet decomposition is to be performed. *Atr-name* should be a triplet-valued attribute. Let $\$A$ be a triplet-valued attribute. The *tdec* operator on $\$A$ creates two extra attributes to accommodate the upper bound and value components of $\$A$. The lower bound component replaces the $\$A$ attribute. These three attributes are renamed as A_l, A_u and A_v to represent the lower bound, upper bound and value components of $\$A$ respectively. Since A_l and A_u represent time points, they are each compressed into 2 bytes. A_l, A_u and A_v are all atomic, with A_v taking the datatype of the value component of $\$A$.

  EXAMPLE :

  /* Tdec emp relation on attribute salary after unpacking salary */

  tdec (unpack emp on salary) on salary

- **NAME** : **TFORM**
  SYNTAX : **tform** rel-name **on** atr-name atr-name atr-name
  DESCRIPTION :
  This operation forms a triplet-valued attribute from three atomic attributes L, U, V. The input relation can be either an existing relation *rel-name* or the output of another extended relational algebra expression *alg-exp*. *Atr-names* are the names of the attributes in the input relation on which the triplet formation is to be performed. *Atr-names* should be in the order lower bound, upper bound and value components

of the triplet to be formed. The resulting attribute becomes triplet-valued.

EXAMPLE :

/* Tform emp relation on attributes $salary_l$ $salary_u$ $salary_v$ */

tform (tdec (unpack emp on salary) on salary) on $salary_l$ $salary_u$ $salary_v$

- **NAME** : **USLICE**
  SYNTAX : **uslice** rel-name atr-name **by** atr-name
  DESCRIPTION :
  Uslice forms the union of the time component of an attribute with respect to that of another attribute. The input relation can be either an existing relation *rel-name* or the output of another extended relational algebra expression *alg-exp*. *Atr-names* should both be triplet-valued. The union of the time components of the two attributes is assigned to the time component of the first attribute. The data types of the attributes remain unchanged. The attribute type of the first attribute is changed to "set-triplet" while that of the second attribute remains unchanged.

  EXAMPLE :

  /* Union slice salary attribute by dname attribute in emp relation */

  uslice (unpack(unpack emp on salary) on dname) salary by dname

- **NAME** : **UNPACK**
  SYNTAX : **unpack** (rel-name/alg-exp) **on** atr-name
  DESCRIPTION :
  This operation unpacks a relation on an attribute. The input relation can be either an existing relation *rel-name* or the output of another extended relational algebra expression *alg-exp*. *Atr-name* is the name of an attribute in the input relation. *Unpack* maps each tuple in *rel-name* into a set of tuples such that each element in the *atr-name* becomes the new *atr-name* value of the resulting tuples, and the tuple component for the rest of attributes are the same as the input tuple if *atr-name* is a nested or a set-triplet attribute. Otherwise, the output relation is just the same as the input relation. *Atr-name* will be an atomic attribute if it was a nested attribute, or a triplet attribute if it was a set-triplet attribute, in the output relation.

  EXAMPLE :

  /* Unpack emp relation on attribute salary (Triplet-valued attribute created) */

  unpack emp on salary

Appendix C

TDBMS INSTALLATION

C.1 Changing the Path Specification of the System

In the present installation, the TDBMS resides in the directory *home/usr3/iqbal/dbms*. The directory configuration is given in Figure C.1. After the system source code is loaded to any directory, the system directory can be set up in any desired way. The files to be adjusted are *path.h* and *Makefile*. These are found under the *src* directory in the system directory of Figure C.1.

(a) Modification of *path.h*: Path.h contains paths for the *users*, *databases*, and the *direct* (*relations* and *attributes* files) file. If the system directory shown in Figure C.1 is utilized, then only the user-name should be changed, else the paths have to be adjusted to reflect the exact position where the respective files can be located. For example, *home/usr3/iqbal/dbms/dbase/sysfiles/users* is the complete path for the *users* file in the system directory.

(b) Modification of *Makefile*: The current path of the directory storing the system executable codes is *home/usr3/iqbal/dbms/bin*. This path can be adjusted to any path specification, i.e *../bin*.

C.2 Adding New Users to the System

The *users* file contains a list of users who are authorized to have access to the system. The *users* file is a text file, with each line having information about each valid user with the following format:

username;capability;dbnames;

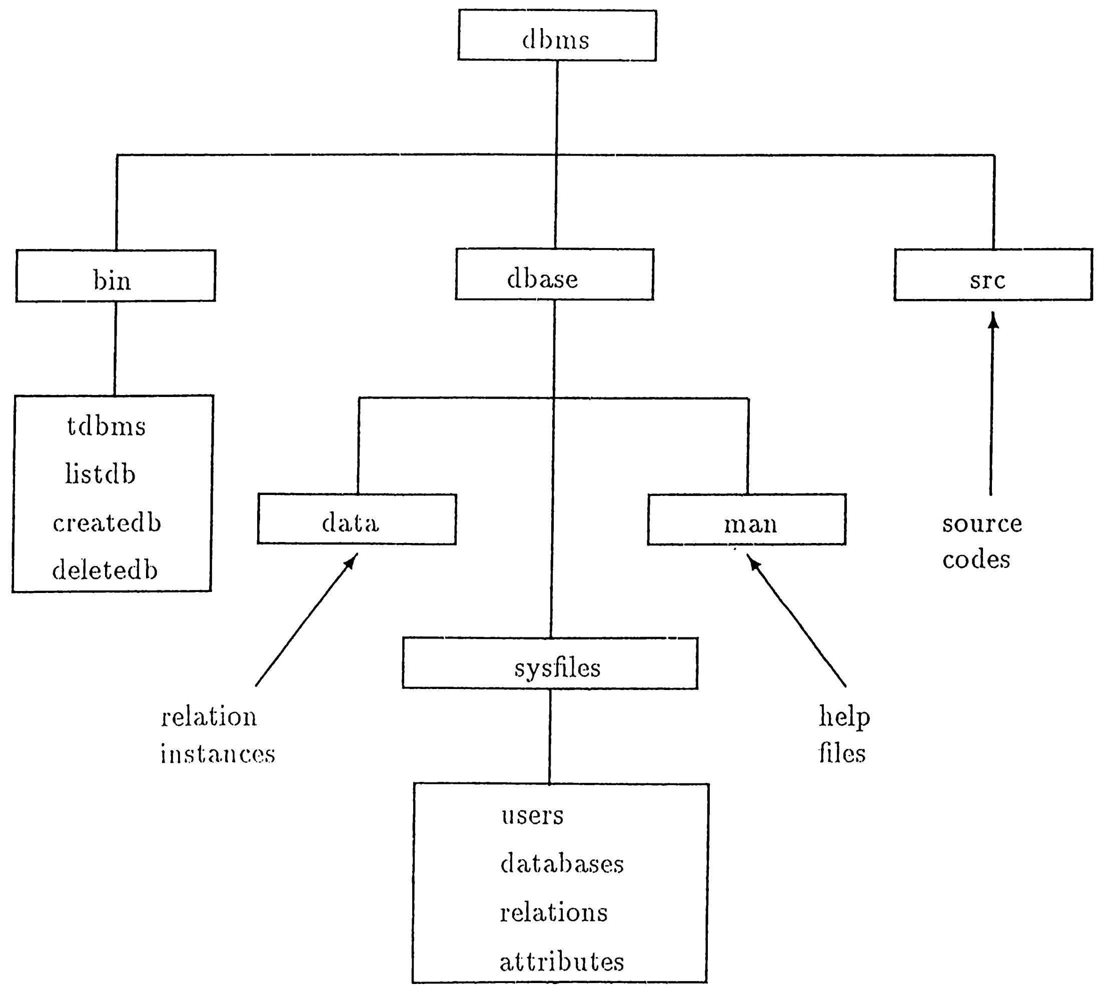


Figure C.1: The Directory Structure of TDBMS System

The user's login name is used for *username*. Capability is **1** for the system administrator and **0** for ordinary users. *dbnames* are the names of databases which can be accessed by this user. *dbnames* are separated by a "," in the file.

C.3 Compiling the System Source Code

The *Makefile* is used for the compilation. It contains all the information about the source files and their dependencies. The first step towards compilation is to change the current directory to the directory where the *Makefile* resides. Next, the MAKE command of UNIX is used as follows:

(a) make (create tdbms executable code)
(b) make createdb (create the createdb executable code)
(c) make listdb (create the listdb executable code)
(d) make deletedb (create the deletedb executable code)
(e) make install (move the executable code to the specified directory, *../bin)*
(f) make clean (remove all the object files)

Appendix D

ALGEBRA EXPRESSIONS FOR THE SAMPLE QUERIES

Note that the relations used in the algebra expressions for **S**, are the current snapshot of the original temporal relations, while those in the algebra expressions for **S/N** are the original relations without the time dimension.

Q1.0 Point Query

S	select salary from emp where ssno = 1
S/N	select salary from emp where ssno = 1
H_C	select salary from (enum1 emp<{ssno} {salary}>{"111111"}) where ssno = 1
H_H	select salary from emp where ssno = 1

Q1.1 Point Query

Same as **Q1.0**, except employee is 250.

Q1.2 Point Query

Same as **Q1.0**, except employee is 500.

Q2 Point Query

S	select ssno from emp where salary > 60
S/N	select ssno from (unpack emp on salary) where salary > 60
H_C	select ssno from (enum1 emp<{ssno} {salary}>{"111111"}) where salary > 60
H_H	select ssno from (unpack emp on salary) where salaryv > 60

Q3	**Range Query**
S	select ssno from emp where salary $\geq$ 50 and salary $\leq$ 70
S/N	select ssno from (unpack emp on salary) where salary $\geq$ 50 and salary $\leq$ 70
H_C	select ssno from ((enum1 emp<{ssno} {salary}>{"111111"}) where salary $\geq$ 50 and salary $\leq$ 70
H_H	select ssno from (unpack emp on salary) where salaryv $\geq$ 50 and salaryv $\leq$ 70
Q4	**Join between two Atomic attributes**
S	select ssno type budget from (proj njoin assigned) where pno = 7
S/N	project ((select pno budget from proj where pno = 7) njoin assigned) on ssno type budget
H_C	project (enum1 ((select pno budget from proj where pno = 7) njoin assigned) <{ssno} {type budget}>{"111111"}) on ssno type budget
H_H	project ((select pno budget from proj where pno = 7) njoin assigned) on ssno type budget
Q5	**Join between two Atomic attributes**
S	select ssno from (proj njoin assigned) where pname = "P7"
S/N	project ((select pno from proj where pname = "P7") njoin assigned) on ssno
H_H	project ((select pno from proj where pname = "P7") njoin assigned) on ssno
Q6	**Join between two Atomic attributes**
S	select type from (emp njoin assigned) where salary = 60
S/N	project ((select ssno from (unpack emp on salary) where salary = 60) njoin assigned) on type
H_C	project ((select ssno from (enum1 emp<ssno salary>"111111") where salary = 60) njoin (enum1 assigned<{ssno} {type}>{"111111"})) on type
H_H	project ((select ssno from (unpack emp on salary) where salaryv = 60) njoin assigned) on type

Q7 Join between two Atomic attributes

S select type from (emp njoin assigned) where salary $\geq$ 60
S/N project ((select ssno from (unpack emp on salary) where salary $\geq$ 60) njoin assigned) on type
H_C project ((select ssno from (enum1 emp<{ssno} {salary}>{"111111"}) where salary $\geq$ 60) njoin (enum1 assigned<{ssno} {type}>{"111111"})) on type
H_H project ((select ssno from (unpack emp on salary) where salaryv $\geq$ 60) njoin assigned) on type

Q8 Join between two set-triplet attributes

S project (dept njoin proj) on pno dno
S/N project (dept njoin proj) on pno dno
H_C project ((enum1 dept<{dno} {budget}>{"111111"}) njoin (enum1 proj<{pno} {budget}>{"111111"})) on dno pno
H_H project (dept njoin proj) on pno dno

Q9 Join between two set-triplet attributes

S : project ((select dname from emp where ssno = 123) njoin dept) on budget
S/N : project ((select dname from (unpack emp on dname) where ssno = 123) njoin dept) on budget
H_C : project ((select dname from (enum1 emp<{ssno} {dname}>{"111111"}) where ssno = 123) njoin (enum1 dept<{dname} {budget}>{"111111"})) on budget
H_H project ((select dname from (unpack (droptime emp on dname) on dname) where ssno = 123) njoin dept) on budget

Q10 Selection on a set-triplet attr., retrieval from another set-triplet attr.

S select salary from emp where dname = "D7"
S/N select salary from (unpack emp on dname) where dname = "D7"
H_C select salary from (enum1 emp<{ssno} {salary dname}>{"111111"}) where dname = "D7"
H_H select salary from (unpack emp on dname) where dnamev = "D7"

Q11 Aggregation (1)

S	emp<{dname} avg(salary)>
S/N	(unpack (unpack emp on salary) on dname)<{dname} avg(salary)>
H_C	(enum1 emp<{ssno} {dname salary}>{"11111"})<{dname} avg(salary)>
H_H	temp1 := droptime emp on dname
	temp2 := unpack temp1 on dname
	temp3 := unpack temp2 on salary
	temp3<{dname} avg(salary)>
H_R	enum2 temp3<{dname} {avg(salary)}>{"010185010186" "010186010187"}

Q12 Aggregation (2)

S	assigned<{ssno} count(pno)>
S/N	assigned<{ssno} count(pno)>
H_H	assigned<{ssno} count(pno)>

Q13 Aggregation (3)

S	dept<{dno} max(budget)>
S/N	(unpack dept on budget)<{dno} max(budget)>
H_C	(enum1 dept<{dno} {budget}>{"111111"})<{dno} max(budget)>
H_H	temp := unpack dept on budget
	temp<{dno} max(budget)>
H_R	enum2 temp<{dno} {max(budget)}>{"010185010186" "010186010187"}

Q14 Aggregation (4)

S	assigned<{ssno} avg(rating)>
S/N	(unpack assigned on rating)<{ssno} avg(rating)>
H_C	(enum1 assigned<{ssno} {rating}>{"111111"})<{ssno} avg(rating)>
H_H	temp := unpack assigned on rating
	temp<{ssno} avg(rating)>
H_R	enum2 temp<{ssno} {avg(rating)}>{"010185010186" "010186010187"}

Q15 Aggregation (5)

S	:	emp<{ssno} max(salary)>
S/N	:	(unpack emp on salary)<{ssno} max(salary)>
H_H	:	temp := unpack emp on salary
		temp<{ssno} max(salary)>