A NEW ACCESS METHOD AND IMPLEMENTATION OF A TEMPORAL WATER RESOURCE DATABASE

By

YUN-CHEN DUNN

Bachelor of Business Administration

National Chengchi University

Taiwan, R. O. C.

1986

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 1991

A NEW ACCESS METHOD AND IMPLEMENTATION

OF A TEMPORAL WATER RESOURCE

DATABASE

Thesis Approved:

Hongbin Liu

Thesis Adviser

J Chandler

D. E. Hokanson

Norman N. Durham

Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express sincere appreciation to Dr. Huizhu Lu for her encouragement and advice throughout my graduate program. I also wish to thank the other members of my committee, Dr. G. E. Hedrick and Dr. J. Chandler, for their suggestions and support during the study.

Many thanks go to individuals who helped me during my staying at Stillwater. Specially, I extend my sincere thanks to Xiwin Chen and her family for their providing a place for me at the final stage of my research. In addition, I would like to thank Mr. Song-shen Yeh for his providing me the computer and a place to stay during the final two weeks of my research. Also, I wish to thank Mr. David Gadi at the Water Research Laboratory, Oklahoma State University, for his assistance in my collecting water data for my project.

Special thanks are due to my family, particularly my father, who encouraged and supported me all the way. My deepest thanks are extended to my husband Song-bin Wu for his moral support, consistent encouragement, and understanding.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The database technique has been applied to many applications. Since a database system employs computers to manage data, it not only saves manpower but also provides an efficient way to either retrieve or store data. Also, a database system can maintain integrity of data, balance conflicting requirements, and allow data to be shared by users. In addition, computers can avoid errors which are made easily by people. Many theories and models of database systems have been proposed and discussed. According to different requests, a database system has been added some other features, such as graphic outputs, except basic functions in many fields. The request of adding the ability to record a time factor has been recognized by many researchers. Many conceptual models have been formulated to provide the ability of handling temporal data in a database system. In addition, it is important to keep historical data in some applications such as decision making systems or water resource management. Conventional databases only keep the current content of a database. Historical data, which is replaced by updating operations, is no longer available to users. A temporal database is designed to handle temporal data efficiently. It records retroactive and

postactive changes as well as the past information of objects in a database. However, since a temporal database requires large amount of storage to store historical data, it has been considered impractical for a long time. Due to the cost of storage having been reduced recently, for instance, optical disks, a temporal database becomes possible and useful. Moreover, more users require other output formats besides a traditional tabular format.

Motivation

A temporal database, which has the ability to record and to process concepts varying with time, and which keeps aged data as well as current data, is employed in many applications. During the past ten years, four types of databases (snapshot, rollback, historical and temporal databases) with differing abilities to support temporal information have been proposed. Most current approaches [LUM84, BEN82 and AHN86] for implementation of temporal databases cannot provide an efficient way to query historical data. Therefore, those approaches are not suitable for applications such as water research, which require both frequent and efficient access to historical data to analyze then report historical statistics.

Furthermore, the main output format of currently available database software is a table which displays an attribute in one column and a record in one row. However, this tabular format is inadequate for applications which need

graphical outputs.

Literature review

Related work about temporal databases and water research is provided in the following sections.

Temporal Databases

Temporal databases have attracted many researchers' attention in the past ten years. Snodgrass [SNOD86a, SNOD85] gave a complete introduction to temporal databases. Coburn [COBU90] discussed problems such as growing storage size and time to duplicate data. He also discussed methods which either change the underlying structure of a database, or add time attributes into a database. He provided an approach which adds two attributes into a database, storing only the modified data. Other related work is summarized in term of the following five aspects.

Model

Some research extends the relational model to include the time factor. Clifford [CLIF85, TANS86] discussed the problems of a temporal database and proposed an approach for an extended relational algebra, which includes five new revised operations: pack, unpack, triplet decomposition, triplet formation, and time slice. He also presented the concepts of a non-first normal form to reduce the redundancy of data. Mckenzie and Snodgrass [MCKE87]

proposed another approach to extend the relational algebra to support transaction time. This approach could apply not only to support of transaction time, but also of valid time in a historical model. A combination of these approaches could yield a temporal algebra. In addition, Ariav [ARIA86] introduced a model, called the Temporally Oriented Data Model (TODM). TODM is a restricted, but consistent, superset of the relational model. Then, he presented TOSQL, a SQL-like query language. Gadia and Yeung[GADI88b] proposed a generalized model of a relational temporal database. They introduced a Boolean algebra of multi-dimensional time stamps. As an application, the same idea could be explained for a two-dimensional model. They also gave a precise way of classifying errors and updating a database.

Segev and Shoshani [SEGE87, SHOS86] introduced another distinct model which both characterized the properties of temporal data and provided operators over them without using the ideas of the traditional model. They presented the concepts in terms of time sequence, a sequence of temporal data for a single entity instance such as the salary history of an individual. Accordingly, the properties of time sequences, such as their type (continuous, discrete, etc), are exploited in order to design efficient physical data structures and access methods for time sequences.

Query Language

A query language extending the relational query language to handle temporal data has been proposed. Snodgrass [SNOD86, SNOD87] developed a new query language, TQuel, to query a temporal database. He provided a tuple relational calculus for the TQuel statements which differ from the corresponding Quel (the query language used in INGRES) counterparts. Finally, he addressed a comparison with other temporal query languages. In addition, Gadia [GADI85, GADI88a] proposed a query language for the temporal database of a homogeneous relational model. Tansel, Arkun, and Orsoyoglu [TANS89] introduced a Time-By-Example (TBE) query language. TBE is a "user-friendly" query language designed for historical relational databases. It uses a graphical structure and the example query concept of Query-By-Example (QBE), and employs the hierarchical arrangement of subqueries of Aggregation-By-Example (ABE) and Summary-Table-By-Example (STBE). In addition to supporting time, it is able to manipulate triplet and set-triplet-valued attributes. TBE adopted an extended relational data model in which a nonfirst normal form and an attribute time stamping are used.

Graphical Query Language

The concepts of a graphical query language have attracted many researchers' attention, since conventional query languages, such as SQL, are difficult to learn for users who are not familiar with the semantics of a query language.

Briefly, a graphical query language is a language using graphics to display a query. Angelaccio [ANGE90] introduced the concepts of QBD*, a graphical query language with recursion. He proposed a system called Query by Diagram* (QBD*) which used a conceptual data model, a query language on this model, together with a graphical user interface to query databases. Consequently, a graphical interface should be formed to provide the graphical query language for a database. Moreover, the ideas of the implementation of a prototype interface between a relational DBMS and interactive computer graphics system were discussed by Spooner [SPO084]. Spooner presented both the database structures used to manage the data and the techniques used to design the interface and discussed an approach to make the interface portable.

Implementation

There are numbers of papers concerning theory and models of temporal databases as have been mentioned previously, but there are fewer papers that provide approaches to the implementation of a temporal database. At first, Ben-Zvi [BEN82] proposed a complete approach of a time relational model. He introduced five time attributes into this model. Then, he presented the overall design and architecture of this model. Ahn [AHN86a, AHN86c] has another approach which concentrates on the access methods and performance analysis. He discussed various access methods, such as reverse chaining, accession lists, indexing, and clustering. He also discussed

the problems of a temporal database; namely, (1) the ever-growing storage size problem and (2) the inefficient conventional access method problem. He claimed that the use of optical disks can overcome the first problem. He proposed an approach of Temporally Partitioned Store (TPS) to solve the second problem. Moreover, an approach designing a DBMS to support a temporal dimension was offered by Lum [LUM84]. Lum addressed the structure, strategy and alternative strategies to support indexing. For instance, he employed two index trees-one for the current index, the other for the history index. If the history index became too large, then a smaller history tree could be created. For example, the history index tree can be separated into history tree #1, history tree #2, etc. Furthermore, he added a pointer into a current data node to point to a future chain to handle future events. He also provided ways for adding effective time into entries in order to correct error entries. Overmyer [OVER82] provided a design and implementation of a time expert for a relational database system.

Physical Storage Organization

One of the main problems in implementing a temporal database is the storage size, which has continual growth to hold old data. Some techniques, such as partition schema of files, and differential files, to overcome this problem were provided by Katz and Rotem [KATZ84, ROTE87].

Furthermore, Ahn [AHN 86b] proposed a method to

evaluate the performance of a temporal database.

Water Resources Databases

The water industry has applied computer techniques to manage data for many years. Before employing computer techniques into the water industry, water resources data were managed manually. For example, water data was recorded on paper by hand. Those people who were responsible for interpreting water data had to work on lots of paper to generate a report. Therefore, they had less time to analyze results and making decisions, which were more important than gathering data. Due to the benefits of employing computer related concepts and techniques, the water industry has obtained many improvements in the management of water data. A committee report [COMP89] outlined the usefulness of computers in the water industry. This report also presented how computers affect the water industry in terms of some current aspects. These aspects are advances in hardware, advances in commercial software, and modeling and planning techniques. It also outlines how computers will be used over the next five years. It gives examples to indicate the importance of applying proper modeling techniques. One researcher, Indira [INDI90], introduced water resources management systems in the environment of Alberta, such as the Water Resources Management Model (WRMM) -- a water resources planning tool, and The Real-Time Data Acquisition System (DACQ) -- an automatic data collection system developed in a

minicomputer environment. She also addressed the fact that the environment of Alberta encounters challenges in managing its water resources information. Some challenges are the following: the inventory of water sources and their characteristics, the large volume and high frequency of data collection, and the request for graphic data representation.

One of the computer techniques which is applied to manage water resources is the database technique. A database is a collection of relations. It provides users not only efficiency in either retrieving or storing data, but facilitates the updating of data also. Chow [CHOW87] gave the usefulness and the requirements of the proper designed water quality database. He also presented examples of their applications in the water industry. He provided the basic design principles: analysis of requirements, conceptual design, predesign, and design for the proper design of the water quality database. Another example of applying the database technique is provided by Mainmone [MAIM89]. This database is designed for application to ground water management. Mainmone also addressed functions and overall resource data included in this system. Actually, this system is a combination of WordStar 2000, Microsoft Chart, and dBase III Plus. Besides, Wright [JEFF85] provided the concepts of rigid format database management systems. He mentioned that a general purpose package, such as Lotus 1-2-3, is difficult to use to satisfy specific requirements and in general, hard to learn. In contrast, custom-made software is

expensive and requires costly maintenance. Therefore, his approach provides a rigid format database in which all the files use the same data structure. Kittridge [KITT86] introduced a water system database for Naples, Florida. This system applies an Intergraph Computer-Aided Design and Drafting (CADD) system to aid the computer modeling of the city of Naples water distribution system. He also compared this database with a traditional manual system. He claimed that CADD provides a system which both efficiently updates out-dated maps and avoids errors produced by manual methods. Furthermore, an example of applying electronic spreadsheets in water resources analysis is provided by Hancock [HANC87]. Hancock mentioned that applying spreadsheets can speed up the parameter estimation process in water resource analysis and provide better control on how these values are chosen. In addition, the WATSTORE database [WATS81], which is used by the U. S. Geological Survey to manage water data files, is a nationwide database. This system consists of several files, such as Ground-Water Site-Inventory File, Daily Values File, and Peak Flow File. The water data stored in the system are grouped into files according to their common characteristics and data collection frequencies. An index file of sites for which data are stored in the system is also maintained by the system.

Motivation for an Implementation of a Temporal Water Resources Database

Because water resources data is usually enormous, many techniques such as Computer Assisted Design/Computer Automated Mapping (CAD/CAM), and geographic data base representations have been employed to manage water resource data [INDI90]. Due to the large volume of water resource data and the need of graphical data representation, more efficient access methods and graphic functions must be added into a water resources database.

Currently, the U. S. Geological Survey (USGS) uses the WATSTORE database which uses a combination of a station name and date as a key to a water file to manage water data. A software package called Hydrodata QW & Water-value [HYDR90] is designed to retrieve data from water files in the WATSTORE database, and provides export formats to associate its data with other packages. However, a new package should be designed for the following reasons.

1. In the WATSTORE database, water data of a station or a lake for a period of time are stored together. Therefore, users cannot obtain water data of a group of stations or lakes for a time point efficiently.
2. The Hydrodata QW & Water-value package has only a retrieving function but lacks an updating function. Therefore, users cannot add a new record into water files.
3. Hydrodata QW & Water-value package provides some export

formats so that their output can be used as an input file to some packages; however, users prefer to work under one package and don't want to switch among packages.

4. It needs more memory storage than the proposed database because associated packages must be resident in the same environment for users to transmit data among them.
5. It lacks graphical outputs, for instance, geographic displays or bar charts.

Therefore, a database which combines graphical outputs, database functions and the abilities of efficiently handling historical data should be designed.

Research Objectives

Because a temporal database applies non-deletion policy [AHN86b], the size of a temporal database is larger than a conventional database. Conventional indexing structures which are designed to access current data are not either suitable or efficient for a temporal database. Although [LUM84, BEN82, AHN86a, b] provide indexing structures, such as reverse chaining and accession lists to access historical data, none of them can retrieve all objects for one time point efficiently. The objectives of this thesis are:

1. to develop an access method for a temporal database which can retrieve all objects in each version of a specific time period efficiently;
2. to analyze access methods in a temporal database;

3. to develop a temporal database for water research with the following properties.
   a. It keeps the historical data of water resources in Oklahoma for the past ten years efficiently.
   b. It provides more features than current packages by adding graphical outputs.
   c. It provides users a easier way to operate the database. (i.e. making a query by moving a cursor within a window(s)).

Organization of The Thesis

This thesis is organized as follows. Chapter I gives the motivation and related work of this research. Basic concepts related to relational systems and temporal databases are provided in chapter II. Chapter III presents the detailed description of the proposed new access method which applies a path copying method. The overall design and structures of the temporal water resources database are described in chapter IV. Chapter V gives the performance evaluation of the proposed method and database. The conclusion and a summary of this thesis are provided in chapter VI.

CHAPTER II

BACKGROUND CONCEPTS

A database system uses computers to keep records for people. Generally speaking, a database system is a computerized record-keeping system. This chapter, first, gives the basic concepts of database systems, such as the components and operations of a database. Then, the concepts of a temporal database which has an ability to process and record temporal information are presented. Four databases according to their abilities to support time dimension are also described in this chapter.

Basic Concepts

Following are the basic concepts of a database system. These concepts are summarized from Date [DATE86].

Models

A database system applies conceptual models, hardware, and software to provide operations for users to access and store data in a database. Three models have been employed in a database system: a relational model, a hierarchic model, and a network model. These models are described briefly as follows.

Relational

A relational system is a system in which the data is viewed by users as tables (relations), and the operations of users (e.g., data retrieval) generate new tables from old ones.

Hierarchic

A hierarchic database can be viewed as set of trees. A tree consists of a single "root" record type, together with an ordered set of zero or more dependent (lower-level) subtree types. Each child record can only have a single parent.

Network

A network system can be regarded as an extended form of the hierachic data structure. The difference is that, in a network structure, a child record may have any number of parents.

Architecture

Three general levels constitute the architecture of a database system: internal, conceptual, and external levels.

Internal Level

The internal level is the closest part to physical storage. It handles the way the data is actually stored. In this level, a file manager is designed to retrieve or store data logically. In other words, the file manager ignores all

details of physical disk of I/O, but uses terms of (logical) "page I/O". On the other hand, a disk manager is designed to handle physical disk I/O. It responds for the request of a file manager by retrieving or storing data from or into physical storage, for examples, disks. In addition, some indexing structures are applied by the database system to facilitate the speed of retrieving data, i.e. B-trees, hashing, and pointer chains.

External Level

The external level is the closest part to the users. It responds to the way the data is viewed by users. There are two kinds of users in a database system, an application programmer and an on-line terminal user. An application programmer can use the data sublanguage, which is embedded within the corresponding host language, to perform database operations. An on-line user can use either a structured query language or a menu-based query language to use operations provided by the system.

Conceptual Level

The conceptual level is the part of indirection between the internal level and the external level. It represents the data differently either to a view of a user or the way the data is actually stored. Broadly speaking, the conceptual level is intended to view data "as it really is".

Relational Data Structure

Most of the current databases are relational. A relational model is easy to understand and manipulate. Usually, a relational database can be viewed as a collection of tables (relations or files). A relation consists of a set of tuples (rows of a table). Each tuple consists of a set of attributes (columns of a table). Each attribute obtains a value from a domain which is a set of atomic values.

Operations

Users can perform a variety of operations on relations in a database: retrieving data from existing files, adding new files into the database, inserting new record into existing files, deleting data from existing files, updating data in existing files, and deleting existing files permanently from the database. A data manipulation language is used to define operations that provide users the abilities to manipulate data in a database. A data definition language is used to define the content of a database.

With different designs, different systems (for example, a temporal database) contain some special features which are not provided by other systems.

Databases Supporting Temporal Information

Although a database provides many benefits to users, some abilities, such as an ability to handle temporal data,

should be added into current databases for some applications. One drawback of current databases is they always keep data in the latest fashion. Operations such as an insertion or a deletion will change the state of a database. Furthermore, it cannot provide information for a query about the past status and retroactive or postactive changes of objects. Since it doesn't store historical data, no trend analysis can be performed under this system. A temporal database which keeps the current data as well as the historical data has been modeled by many researchers in the past ten years. In order to describe the properties of databases which can process temporal data, Snograss [SNOD85, SNOD86a] provided a new taxonomy of time for use in a database. According to the abilities of presenting the temporal information, four types of databases are described as follows.

Snapshot Database

Conventional databases always provide the current state of a database. Any updating operation, such as either inserting a new record into a relation or deleting an existing record from a relation, moves the database to a new state. Moreover, after each updating, the old data is lost totally. Users can no longer know what has been stored in the past. This type of database is termed a snapshot database. In the relational model, a database is viewed as a collection of relations (files). A relation is usually represented in a two-dimensional table.

Since this kind of database doesn't provide the ability to keep historical data, users cannot query the past state of this database. For example, an employee relation of a time point may look like this.

name	salary
Mary	20k
Harry	30k

Figure 1. A Snapshot Relation

The following query cannot be answered under a snapshot database:

What was Mary's salary last month ?

Did Harry earn more money than Mary last year? Users can only get that Mary's salary is 20K and Harry's salary is 30K, right now. Therefore, a snapshot database is obviously inadequate in many applications. Without system support, many applications have to handle historical data in an ad hoc manner.

Rollback Database

One solution to the above deficiencies is to store all past states of a database, indexed by the transaction time. A transaction time is when a record enters the

database. It doesn't need to reflect when a record is valid. The answer to the query about the past state of a database can be obtained by rolling back the database to a certain time point. Therefore, this operation is termed rollback. A database which supports this operation is termed a rollback database. A rollback relation is shown in Figure 2.

In the rollback database, we can obtain the answer to the following query:

What was Mary's salary last month?

But, there is no way to record retroactive or postactive changes and correct errors in past tuples.

name	salary	transaction time (begin)	(end)
Mary	15K	01/25/80	07/31/83
Mary	20K	08/01/83	-
Harry	30K	12/07/82	-

Figure 2. A Rollback Relation

Historical Database

While a rollback database stores a sequence of static states, historical databases record a single historical state per relation as it is best known. When errors are

discovered, the database is modified by moving back to an expected time point to correct errors. Since previous states are not retained, there is no way to view the database as it was in the past. Historical databases use valid time to be a time axis.

The same relation in Figure 1 may appear as indicated below in a historical database.

name	salary	valid time begin	end
Mary	15K	08/01/81	01/15/88
Mary	20K	01/15/88	-
Harry	30K	09/10/82	-

Figure 3. A Historical Relation

A historical database uses valid time as an axis to a relation. Therefore, it more closely resembles the real world. However, any error correction for a past tuple causes the original status of this database to change.

Temporal Database

A temporal database supports both transaction time and valid time. While a rollback database views stored records as

some moment of time, a historical database views them as being as valid as some moment of the present. A temporal database views stored records as being valid in some moment of some time. A temporal relation may be thought of as a sequence of historical states, each of which is a complete historical relation, indexed by the transaction time. The weakness of a temporal database is its requirement of a large amount of storage to store the historical data. A temporal relation is illustrated in Figure 4.

name	salary	valid time		transaction time	
		begin	end	begin	end
Mary	15K	08/12/82	01/25/88	09/01/82	01/31/88
Mary	20K	01/26/88	-	01/31/88	-
Harry	30K	07/25/82	-	07/20/82	-

Figure 4. A Temporal Relation

CHAPTER III

A NEW ACCESS METHOD USING A MODIFIED PERSISTENT B-TREE

Other Access Methods

The inefficiency of conventional indexing structures for databases which support a time dimension have been recognized by researchers. Approaches which provide indexing structures for databases to handle the time factor are discussed below.

Lum's Approach

Lum's approach creates two index trees to manage the current and the historical data separately. The current index tree stores all nodes which point to all current tuples. The history index tree stores nodes which are deleted from the database. When an object is deleted by request, a node representing this object is moved from the current index tree to the history index tree. Historical data of each object chain together in a time decreasing order. If an object is alive, a node in the current index tree will point to the head of the historical data. Otherwise, a node in the history index tree will point to the first tuple of the historical data. A query to retrieve a deleted object of a time point

can be answered by searching the current index tree, first. If a node cannot be found in the current index tree, searching is applied to the history index tree. Then, a sequential scan is performed to obtain the expected tuple. In addition, Lum's approach stores all tuples together.

Ben-Zvi's Approach

Ben-Zvi's approach applies one index B+ -tree to manage data. He separates storage into two groups: the current group and the history group. The current group provides a current page to store all current tuples. The history group employs history pages to store history tuples. The historical tuples of an object are chained together to form the tuple-history chain. The index B+ - tree keeps TIDs to each current tuple in the current page. The current tuple maintains a pointer pointing to the tuple-history chain, which uses the current tuple as a head in the history pages. Therefore, the current tuple is kept both in the current page and the history page.

Ben-Zvi further applied a history TID in the index tree to point to the tuple-history chain directly. Using a history TID can avoid the deleted current tuples from remaining in the current page. Figure 5 illustrates Ben-Zvi's approach.

Ahn's Approach

Ahn's approach provides some methods to handle temporal information. He applies the temporally partitioned storage structure to separate the current data from the historical

data. Then, some mechanisms are provided to manage the current data and historical data individually. Since the current data is accessed frequently, any access mechanism used for the primary store can be applied to handle current data efficiently. Then, some access techniques are applied to maintain the historical data. They are described as follows.

Reverse Chaining

A reverse chaining method links all historical data of an object in a reverse order starting from the current tuple. When a tuple is replaced, the current tuple is moved to the history store. Then, a new tuple is inserted in the current store and keeps a pointer to the predecessor that was moved to the history store.

Accession Lists

If the historical data of an object keeps growing, the length of the historical chain grows long. It may be too slow to traverse the chain in order to retrieve data. One way to solve this problem is to maintain accession lists between the current store and the history store. The accession list is a full index of the historical data of the corresponding object. Using this mechanism can reduce the time needed to retrieve data from the history store.

Other mechanisms (Clustering, Stacked Versions, and Cellular Chaining) provide techniques to group historical data together. After applying these methods, access time

to the history store can be reduced.

A Modified Persistent Search B-tree

This section gives the basic concepts of a persistent search tree. Then, definitions, algorithms, and protocols for a modified persistent B-tree are provided in the following sections.

Basic Concepts

The persistent search tree has been applied to many fields recently. This structure creates a new version of the tree after either an insertion or a deletion, but an old version still can be accessed. There are two ways to make a search tree persistent: path copying and node copying. Path copying can work on any kind of tree. Sarnak and Tarjan provided algorithm for red-black trees [SARN86], Mayers used AVL trees, Krijnen and Meertens used B-trees, and Reps, Teitelbaum, and Demers employed 2,3 trees. Node copying was first addressed by Sarnak and Tarjan to create space-efficient persistent search trees. Moreover, Kazerouni-Zand [KAZE88] provided a persistent B-tree algorithm which is similar to the limit node copying method [SARN86]. Since the persistent B-tree keeps paths to each time versions, it is suitable to be the indexing structure in a temporal database. The main drawbacks of a persistent B-tree which uses a path copying method are its nonlinear space requirements [SARN86] and the waste nodes along the path

[KAZE89]. Consequently, Sarnak and Tarjan proposed a node copying method which adds an auxiliary pointer list in a node to point to the time stamp of each child. This method overcomes the former problems, but it increases the time complexity since searching the pointer list to find a proper pointer is time consuming. Then, they provided one other method: limit node copying which adds p slots for time pointers in each node. The brief description of three methods is as follows.

Path Copying

This method, first, copies nodes in which changes are made. Then, any node that contains a pointer to a node which is copied must itself be copied. This means that a path copying method will copy the entire path from the root to the node. Any node which is not along path will not be changed. This method is different from copying an entire tree, since the new version shares nodes, which are not on path, with the old version.

No Node Copying

A no node copying method maintains a list of pointers in a node which point to the time stamp of each child. No node will be copied by using this method. Since each updating may cause a pointer to be added into the list of pointers, the list can grow arbitrarily large. The no node copying method saves space compared to path copying; however, it increases

time complexity for searching for the pointer of each child.

Limit Node Copying

This method keeps p slots for time pointers instead of the unlimited size of time pointers. A node is copied when there is no free slot for the coming time pointer.

Definitions

Major definitions used to describe operations for manipulating a persistent B-tree are defined below.

RETRIEVAL (KEY, t): A retrieval operation retrieves the key of an object from the database version with time-point equal to t. The time point, t, is defined as follows.

```
If t >= current_time_point  then
     t = current_time_point
else  if t < current_time_point then
          t = the available time-point which is the
               closest time_point to t
```

The time point, t, for insertion, deletion, and updating operations is defined below.

```
If t > current_time_point then
     copyflag = 1
     curr_time_point = t
     create_new_time_root (t)
else  if t < current_time_point then
        return (error_message: invalid time point)
```

INSERTION (KEY, t): An insertion operation inserts the key of an object into a database version with time-point t using the path copying technique.

DELETION (KEY, t): A deletion operation creates a version with time-point equal to t. This version excludes the key of the deleted object.

UPDATING (KEY, t): An updating operation creates a new time version, t, which consists of the updated data of the object with the key, KEY.

Algorithms

In this thesis, a new access method for a temporal database is developed. This new access method applies Sarnak's path copying algorithm to a B-tree with persistent concepts [SARN86]. Following are algorithms for searching, insertion, deletion, and updating operations for a persistent B-tree.

Searching

A searching operation is used to find a proper location of the key value, KEY, before insertion, deletion, or updating operations are applied. In searching, the "time-stamp-root B-tree" is searched first to find the root with time-point t. Then, searching the value, KEY, proceeds through the whole tree which is rooted by the found root. If the key is found, the position of this key is returned.

Otherwise, the next position of the largest key which is smaller than KEY is returned.

Insertion

At first, a search operation is applied to find the location for the key being inserted. If the copyflag is on (i.e. copyflag = 1), the nodes along the path are copied. The nearest node which has a free slot is marked as a star node during searching. Then, top-down insertion is performed as follows. If there is a free slot in the node where the key is inserted, the key is inserted directly. Otherwise, a split operation is applied to the node and promoting a key to an upper level will be performed until the star node is met. Figure 6 gives an example to illustrate how this insertion operation proceeds.

Deletion

The deletion operation uses a search operation to find the location of the key, KEY, which should be deleted. If copyflag is on, the nodes along the path are copied. The star node is also marked during searching. Then, the key, KEY, is removed from the node, if the KEY is found. Otherwise, an error message is returned. After deletion, if underflow happens, merge or redistribute operations proceed. An example of a deletion operation is shown in Figure 7.

Update

An update operation is performed with the path copying technique. Because we need to keep the key, KEY, in the old node, the same key value is inserted into a new node with new offset (an offset is a record number). The new offset keeps the address of a record where the changed data is stored. This method is similar to an insertion operation. The only difference is that the same key value exists in both the old and the new nodes, but the offset of the key is different. When values in attributes of a tuple change very often in a database, an updating operation will be used very often.

Node Format

The sample nodes of a persistent search B-tree and a "time-stamp-root B-tree" are in formats as shown in figure 8 (a) and (b). An internal node keeps pointers to its child (s) (i.e. the "pointer" field of the sample node stores these pointers). A leaf node stores an address where a real record is stored. This address is kept in the "pointer" field of a leaf node. The KEY field of the sample node stores the primary key value of a persistent B-tree.

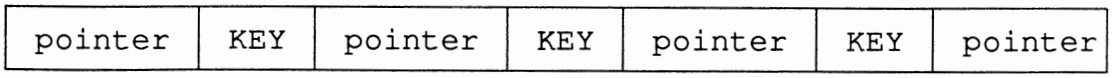
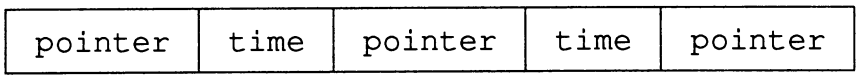


Figure 8. Example of Nodes

a. A Node Example of a Persistent B-tree



b. A Node Example of a "Time-Stamp-Root B-tree"

Figure 8. (Continued)

Protocols

Protocols for search, insertion, deletion, and update operations for a persistent B-tree are based on the B-tree with persistent concepts. A data structure, PAGE, represents a node of a persistent B-tree. KEY is the key which is searched by request. Found_RRN represents the record number of the found PAGE. Found_POS is the position of a searched key in a node. Promo_R_child is the record number of NEW PAGE which is created by a split operation. Promo_KEY is a key which is promoted from the lower level.

```
struct BT {
 int keyct;              /* number of keys in the node.  */
 char key [maxkeys][maxkeylen + 1];     /* key values.  */
 long child [maxkeys+1]; /* pointers to each child.*/
 long parent_node;       /* parent node.  */
} PAGE;
```

current_time_point: Time indicates the last operation.

copyflag: A copyflag indicates if a new version is going to be created by path copying.

root_time_stamp: The root of the "time_stamp_root B-tree".

NEWPAGE: A new page created by a split operation.

read_node(): Reading a specified PAGE.

create_time_root(): Inserting a time stamp into a "time_stamp_root B-tree".

Search

A search operation calls subsearch to find the corresponding time stamp in a "time-stamp-root-B-tree" and an object with the key value, KEY, in a subtree which is rooted by the found time stamp. Figure 9 illustrates the protocol of a search.

```
SEARCH (KEY, t, found_RRN, found_POS)
  begin
      root = root_time_stamp      /* find a time stamp  */
      found = subsearch (root, t, found_RRN, found_POS)
      if ( not found ) then
         return (error_message: invalid time point)
      else
         offset = leaflevel (found_RRN)
         root = read_node (offset)  /* get an expected
                                       time stamp        */
                                    /* search the KEY    */
                                    /* in a PB-tree      */
         found = subsearch (root, KEY, found_RRN,
           found_POS)
         if (found) then
            return found_POS
         else
            return 0
end

subsearch (RRN, KEY, found_RRN, found_POS)
 begin
        read the record at RRN into PAGE
        if PAGE is in a leaflevel then
```

Figure 9. The Protocol of a Search

```
            found_RRN = RRN
            search KEY along PAGE
            if KEY is not found then
               found_POS = the POS of a largest key
                           which is smaller than KEY + 1
               return "not found"
            else
               found_POS = POS
               return found
         else
           if there is a free space in PAGE then
              set star_node = RRN
              search along PAGE for KEY
              if KEY is found then
                found_RRN = RRN
                found_POS = POS
                return "found"
           else
                return (subsearch (PAGE.child [POS], KEY,
                          found_RRN, found_POS))
end
```

Figure 9. (Continued)

Insertion

An insertion operation inserts the key value, KEY, of an object into a B-tree with time stamp t. If a copyflag is on, a new database version with time stamp equal to t is created. The protocol of an insertion is provided in figure 10.

```
INSERTION (KEY, t)
 begin
       if  t > current_time_point  then
            create_time_root (root_time_stamp, t)
            copyflag =  ON
       found = SEARCH (KEY, t, found_RRN, found_POS)
       if (found)  then
          return (error_message: duplicate key)
       else
```

Figure 10. The Protocol of an Insertion

```
          if (copyflag) then
             copy nodes along the path
          read the record at found_RRN into PAGE
          insert_promotion (found_RRN, found_POS, KEY)
 end


insert_promotion (POS_RRN, promo_R_child, promo_key)
  begin
        read the record at POS_RRN into PAGE
        if there is a space in PAGE then
           insert promo_key, promo_r_child into PAGE
        else
           P_B_key = promo_key
           P_B_RRN = promo_r_child
           split (P_B_key, P_B_RRN, PAGE, promo_key,
           promo_R_child, NEWPAGE)
           write PAGE to file at POS_RRN
           write NEWPAGE to file at prom_R_child
        if PAGE is not the star_node then
           insert_promotion (PAGE.parent_node,
              promo_R_child, promo_key)
   end

split (KEY, RRN, PAGE, promo_key, promo_R_child,
         NEWPAGE )
     begin
        copy all keys and pointers from PAGE into a working
            page that can hold extra key and child.
        insert KEY AND RRN into their proper places in
            the working page.
        allocate and initialize a new page into the B-tree
            file to hold NEWPAGE.
        set promo_key to value of middle key, which will be
            promoted after the split.
        set promo_R_child to RRN OF NEWPAGE.
        move keys and child pointers < promo_key from the
            working page to PAGE
        move keys and child pointers > promo_key
          from the working page to NEWPAGE.
     end
```

Figure 10. (Continued)

Deletion

A deletion operation deletes the key value of an object at a time point t. If copyflag is on, a new database version

with time stamp equal to t is created. The protocol of a deletion is shown in figure 11.

```
DELETION (KEY, t)
    begin
        if t > current_time_point  then
           create_time_root (root_time_stamp, t)
           copyflag = ON
        found = SEARCH (KEY, t, found_RRN, found_POS)
        if (not found) then
           return (error_message: key not exist)
        if (copyflag) then
           copy nodes along the path
        read the record at found_RRN into PAGE
        if PAGE is in a leaflevel then
           delete KEY from PAGE
           if keyct < m/2  then     /* underflow occurs */
              if sibling node (s) has keys > m/2 then
                 redistribute (PAGE, sibling nodes)
              else
                 merge (PAGE, sibling nodes)
end

redistribute (PAGE, sibling nodes)
 begin
        keyct = keyct of PAGE + keyct of a sibling PAGE
        promo_key = key [keyct/2]
        if redistribute with right sibling PAGE  then
           move keys in sibling PAGE < promo_key into PAGE
        if redistribute with left sibling PAGE then
           move keys in sibling PAGE > promo_key into PAGE
        switch promo_key with parent_key
 end

merge (PAGE, sibling PAGES)
 begin
           move parent_key and sibling keys into PAGE
 end
```

Figure 11. The Protocol of a Deletion

Update

An update operation updates the data of an object with

the key value, KEY, at time point t. If copyflag is on, a new time version is created. Figure 12 gives the protocol of an update.

```
UPDATE (KEY, t)
  begin
       if  t > current_time_point  then
            create_time_root (root_time_stamp, t)
            copyflag =  ON
       found = SEARCH (KEY, t, found_RRN, found_POS)
       if (not found)  then
          delete_path
          return (error_message: key not exist)
       read the record at found_RRN into PAGE
       RRN = leaflevel (found_POS)
       move new record into PAGE
       write PAGE into file at RRN
end
```

Figure 12. The Protocol of an Update

CHAPTER IV

IMPLEMENTATION OF A TEMPORAL WATER RESOURCE DATABASE

This chapter gives the detailed description of an implementation of a temporal water resources database. This temporal database builds its indexing structure by applying those concepts which are discussed in the former chapter. Also, the components of this database system and their detailed functions are presented. Then, the input data and output of this system are described at the end of this chapter. The C programs of implementation of this database apply algorithms and function libraries from Stevens A. [STEV87a, b] and Stevens R.T. [STEV89].

Components of the Database

A diagram of the overall architecture of this temporal database is shown in Figure 13.

Database Schema

A database schema describes the contents of a database and their definitions. For example, a database schema provides information such as the data attributes of each relation in the system and the primary key of each relation. Since a database schema provides the skeleton of a database,

it is very useful when a database is modified.

Sliding Bar Menu (SBM)

This Sliding Bar Menu located on the top of the screen contains all categories of services provided by this system. When a cursor moves into a category, a window will be created to show a list of functions. The Sliding Bar Menu not only helps users to select choices more easily but provides an overview of each category. The detailed description of the SBM is provided in the following pages. A procedure to specify a query and a description of the hardware for graphic display are also presented in this chapter. The contents of a Sliding Bar Menu is shown in Figure 14. The contents of a subwindow of each category are illustrated in Figure 15.

Functions of the Sliding Bar Menu are described below in detail.

Setup/quit

This operation initializes the environment that this system requires. After applying the setup operation, an empty database is created. Also, the quit operation provides an exit function for users to exit from this system.

Create

This category creates a view for users. A view, a virtual table, can be created and saved through this operation. There are two functions in this category: save

and retrieve. The save function saves the current view for users. The retrieve function retrieves an existing view for users.

Manipulate

This category provides on-line retrieval and updating functions. Users can modify a database through updating facilities (inserting, deleting, updating). Therefore, they can build their own relations by employing those facilities flexibly.

Retrieve. Users can retrieve data from a database by specifying a query via this retrieval facility. A query can be created by moving a cursor within a window. Moreover, a conditional box is designed for users to specify a query by setting a boolean predicate.

Inserting. A new entity or object can be inserted into a relation via an inserting operation.

Deleting. An existing entity or object can be deleted by way of a deleting operation.

Updating. Modifying data of an object of a relation is performed by an updating operation.

Op/cls

This function provides users the ability to open or close a relation or view.

Display

Four kinds of display formats are provided by this system: geography, diagram, table, and bar chart. Each one has its special properties and advantages for illustration. These formats make output of a query more meaningful and comprehensible than those of other database software. The detailed description of each format is provided later.

Utilities

Four utilities provided by this system are described below.

Math Functions. Math functions, such as sum, max, min, mean, and differential, can be applied to all or some attributes of a relation. For example, a differential function can be applied to an attribute of a relation of water quality such as chemical level. Then, if the chemical level of a station exceeds a standard, this reveals that the water quality of this station is in danger.

Database Size Calculator. The size of a database can be calculated and reported via this function.

System Catalogue. A system catalogue stores information about this system such as attributes of a relation.

File Reorganizer. Water files of the WATSTORE database can be reorganized to be an input file to this database via the file reorganizer. Then, data which are collected by local

researchers and the U. S. Geological Survey can be both stored in the proposed database.

Specification of Queries

Some methods have been proposed to specify a query for a database, for instance, a structured query language, a QBE query language, a TBE query language, and a menu query. The proposed database applies the menu query method. This menu query makes it easier for users to specify a query without knowing the underlying structure of a database. However, a structured query language requires users to learn its semantic structure first before they use it.

Output Display

Four kinds of display formats: geography, diagram, table, and bar or line chart, are described as follows.

Geography

Geographical output is performed by first displaying the shape of Oklahoma on screen. Then, results of a query are illustrated on the map. For example, if users need to know the silver levels of stations on rivers in Oklahoma in 1985, the screen first displays the shape of Oklahoma. Then, varied silver levels are indicated by the different colors of stations on rivers. Therefore, users can get the whole situation of silver level of rivers in Oklahoma through geographical display in less time than that needed by a

table display. Some advantages of geographical display are as follows.

First, it enhances users' geographical concepts: for example, geographical display gives users geographical locations of rivers and lakes in Oklahoma.

Second, it directly attracts users' attention: users can immediately recognize the difference between rivers or lakes via the graphical display.

Third, it provides clustering information: for example, the different amount of rainfall can cluster rivers and lakes into different levels which are indicated by different colors.

Fourth, it presents a warning function: for example, when the value of an attribute is compared to a standard, the alert red color can appear on rivers or lakes to indicate that values are over the standard.

Fifth, it provides the concepts of coexistence: coexistence is important when users analyze the relationship between rivers or lakes.

Diagram

A diagram is a useful tool for expressing relationships among objects. This system provides a dependency diagram to illustrate upstreams and downstreams locations of objects. For instance, a diagram illustrates upstream stations and downstream stations of a query station.

Table

A two-dimensional table is displayed, in which selected attributes appear on the top of columns. The tabular format can represent a very large amount of data.

Bar Chart

A bar chart illustrates the variation of values in an attribute of a relation for an object in a time period.

Line Chart

A line chart format is suitable for a trend analysis, since it can attractively illustrate the result of a trend analysis.

Hardware for Display

A graphical interface is designed for performing graphics outputs. This interface applies the concepts proposed by Spooner [SPO084]. It consists of two components which are described below. A component diagram is shown in Figure 16.

Graphics Interpreter

A graphic interpreter provides two functions: interpretation and mapping. An interpretive function extracts the graphic key of each tuple of the result of a query. Then, a mapping function maps the graphic key to a graphic symbol in a graphic symbol pool and displays the graphic symbol on the

screen. This interpreter creates a semantic relation to store information about mapping from a graphic key to a graphic symbol.

Graphics System

A graphics system stores modules for drawing each graphic symbol.

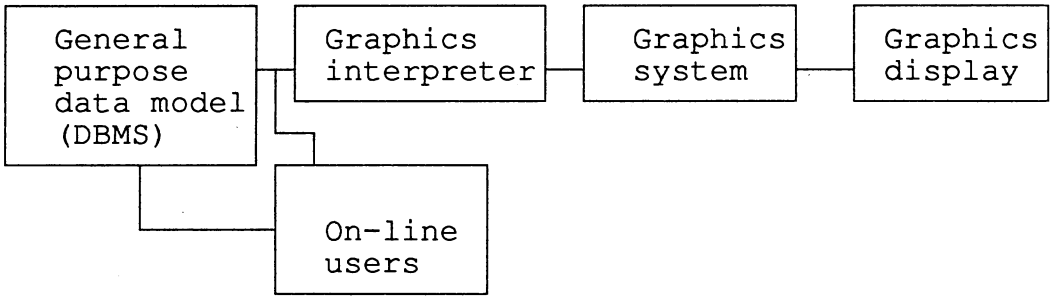


Figure 16. A Component Diagram for the Graphic Interface

Database Management System (DBMS)

The DBMS is an interface between users and the physical organization. It collects menu queries which users request. Then the DBMS, according to the requirements of the query, performs corresponding actions. The DBMS reorganizes water files of the WATSOTRE database to produces some separate relations. The DBMS also modifies the underlying data structure of a database to handle temporal data instead of adding some time attributes into a database.

Storage Subsystem

In the logical structure, this system provides a data file manager to handle all accesses to data files and an index file manager to handle the loading and saving of an index file. In the physical structure, the data files of the same time version will be stored in the same page, and an index of a relation is stored in the same file. The indexing structure of this database employs the proposed method: a persistent B-tree is built for each key of a relation--a primary key and secondary keys. A B-tree is created to manage time stamps of a persistent B-tree. Therefore, the access time to each time stamp can be reduced.

Storage Schema

In addition to an efficient indexing structure for a temporal database, we also must design a storage schema to store both current data and historical data. In order to access all objects in a time version efficiently, storing data of the same time version together can reduce access time. Moreover, a historical chain can be achieved by adding a pointer to different time versions of a tuple. An example of this schema is illustrated in Figure 17.

Input Data

The input data for this system is either reorganized

from water data of the WATSTORE database or entered by on-line users. Two input templates are provided to help users enter data into this database. Figure 18 gives these formats. The entry templates show a "prompt" to ask users to key in input data. If this transaction is committed, data will be put into this database. Otherwise, error messages will be shown on an error message window. Also, two relations are stored in this system--waterdat and chemical. Other templates are also provided by this system to help users make selections, such as a time range box or a relation listing box.

Output

When a query is specified by a user, the result of this query is put into a file called view.dat. A user can save this view for later retrieval by making a selection under the create category. In addition, those results can be output through formats which are available in the system--geography, diagram, table, and chart. Examples of graphic outputs are provided in figures 19, 20, 21, 22. Those outputs are hardcopies of the implemented program.

Geography

This format, first, shows a shape of Oklahoma on the screen. Then, objects are illustrated in the corresponding places in the screen according to their locations. A small box, which is colored, is used to represent each object. Those

colors indicate levels those objects belong to. This output format not only provides the location of an object but attracts users' attention. Figure 19 gives this output format.

Diagram

Relationships among objects are important in water resources. Sometimes coexistence can explain the causes of problems. A dependent diagram is provided by this system. When a user selects this output format, the stations on the upstream and the downstream of the requested object are shown on the screen. An example of this output format is shown in figure 20.

Table

This is a common output format of a database system. This format displays the result in a table. In addition to the tuples of the result, some math information can also be calculated and reported, such as min, max, and average values. An example of this output format is shown in Table 3.

Chart

A line chart is used to display variation of an object in a time period. It is especially useful for a trend analysis. Figure 21 illustrates an output of a bar chart. Figure 22 shows an example of a line chart.

CHAPTER V

PERFORMANCE ANALYSES

This chapter, first, discusses the time and space complexity of a modified persistent B-tree. Then, a performance comparison with three other approaches is presented. The advantages over other databases and comparison with other databases also are addressed.

Time and Space Complexity of the Proposed Method

The time complexity of an insertion or a deletion in a persistent search B-tree is $O(\log_{m1} k + \log_{m} n)$, where k is the number of nodes in a time-stamp-root B-tree, m1 is the order of a time-stamp-root B-tree, m is the order of a B-tree and n is the number of nodes in a B-tree. $\text{Log}_{m1} k$ counts the time for finding a corresponding time stamp in a time-stamp-root-B-tree. $\text{Log}_{m} n$ accounts for finding an expected object in a subtree which is rooted by the found time stamp. For each updating operation, the time complexity is the same as that for an insertion or a deletion.

For retrieving historical data of an object, the time complexity is $O(\log_{m1} k + \log_{m} n + \log n1)$, where n is the number of nodes in a B-tree, m is the order of a B-tree and n1 is the number of nodes in the Beginning Time Binary

Tree (BTBT) which is a binary tree used to indicate when an object first was inserted. At first, it takes O(log n1) time to find the beginning time stamp. Then, an expected object is searched for in the subtree rooted by the indicated time stamp. Therefore, $O(\log_{m1} k + \log_{m} n)$ should be taken into account. As a result, the total time complexity for retrieving a historical datum is $O(\log n1 + \log_{m1} k + \log_{m} n)$. It takes $O(\log k + n \log n)$ time to retrieve all objects in a given time version. Because retrieving an object in a time version takes $O(\log_{m} n)$, retrieving n objects will take $O(n \log_{m} n)$.

Since we apply a path copying technique, it takes $O(\log_{m} n)$ space per insertion, deletion, or update operation.

A Performance Comparison of The Proposed Approach with Other Approaches

After analyzing the performance of the proposed approach, the author compares the proposed approach with other approaches ([LUM84, BEN82, AHN86a, c]) as follows.

Lum [LUM84] introduced an approach which creates two indices, one for current data and the other for historical data (see Figure 23). This approach achieves the basic request for an ability to access both current and historical data. Its performance is worse than the proposed method since its time complexity increases due to Lum's separate indexing structure. When a tuple is deleted in Lum's approach, the key node is removed from the current index tree to the history

index tree. Therefore, if a user wants to retrieve this deleted tuple, searching first proceeds on the current index tree. Then searching works on the history index tree. As a result, it takes O(k + log n2 + log n) time to search for an object with a time stamp, where k is the number of time stamps, n is the number of nodes in the current index tree and n2 is the number of nodes in the history index tree. Moreover, it takes O(n(k + log n2 + log n)) time to retrieve all objects in one time version. The space complexity of Lum's approach is O(1) per insertion and deletion. Figure 23 shows this approach.

A better approach was proposed by Ben-Zvi. It creates only one index tree which keeps the current TID (tuple identification) of a current tuple or object in the current page. This current tuple keeps a pointer to point to the beginning of a historical chain in history pages. Therefore, the time complexity of searching for an object is $O(k + \log_m n)$, where k is the number of time stamps, m is the order of B-tree, n is the number of nodes in a B-tree. Because this approach applies a sequential scan to find an object in a historical chain, a O(k) should be taken into account. The complexity of retrieving all objects in one time version is $O(n(k + \log_m n))$. However, this approach requires more space for keeping a current tuple, both in the current page and in history page.

Ahn [AHN86a, c] provided an approach which improved on Ben-Zvi's method. He provided access mechanisms such as

the accession list, to reduce the time for scanning the historical chain of an object. The accession list mechanism uses the current tuple as the head of a historical chain of an object. Then he added a time-stamp list between the current and historical storage (see Fig. 20). This method uses larger space than our proposed method, since Ahn's approach requires a complete and separate record file for each time stamp. In our proposed method, a path copying technique provides record sharing if the record (node) does not change with time. A summary of the time complexity of each approach is listed in table 2.

Advantages of This Proposed Temporal Database

This section provides the advantages of this proposed database over the Hydrodata QW & water-value. Also, a comparison of the proposed database with other database management systems (WATSOTRE, INGRES, DB2, dBase III plus, and Water Data Information System (WDIS)) is presented at the end this section.

Advantages Over the Hydrodata QW & Water-value Package

Advantages of the proposed database over the Hydrodata QW & water-value is summarized below.

First, updating facilities are added into this database system. Therefore, local water researchers (system users) can

add data they collect into the system.

Second, it provides graphical facilities. Then users can capture more ideas about water data through those graphical outputs.

Third, it provides a conditional box for users to retrieve specific data. Therefore users can get more options for setting a Boolean predicate. For example, a user can enter some attribute's name or number in a conditional box. The result of this query contains only those expected attributes. A simple query can be quickly answered.

A Comparison With Other Database Management Systems

The proposed database is designed for a single user. It provides a menu query, graphical outputs, and an ability to handle temporal data, while other databases don't include all these features.

The indexing structure of the WATSTORE database, which uses a composite key with name and date as a primary key to access a water file, uses a technique similar to the accession list of Ahn's approach [AHN86a, c]. This structure takes more time to retrieve all objects in one time version than that of the proposed database (Detailed analysis is provided in chapter IV). Furthermore, since the WATSTORE database stores all current and historic data together, it will penalize users who make a query to the current version with longer retrieving time than that using proposed database. Both INGRES and DB2

are designed as multi-user systems. They have abilities to process large amounts of data. However, they are useful for the technical users (users who have knowledge of database languages e.g. SQL), but not for the non-professional users. On the other hand, dBase III plus is designed for a single user. It provides a menu query and SQL-like programming language, but it lacks both graphical outputs and the ability to handle temporal data. The Water Data Information System (California Water Database, Mervine & Pallesen Inc. [MERV90]) is designed to promote consistency among local water districts, state users, and Federal agencies, for instance, the USGS. The Water Data Information System (WDIS) is built with the INGRES relational database management system. In addition, it runs on a distributed network so that state users or Federal agencies can access or update data in this system concurrently. Therefore, the WDIS can not only enhance consistency of water data, but also decreases the isolation of agencies. However, none of the databases provide both the graphical outputs and the ability to handle temporal data. The summary of the comparison is shown in Table 2.

CHAPTER VI

SUMMARY AND CONCLUSIONS

Many applications of databases must keep the old data as well as the current data. Consequently, according to the methods of supporting the time dimension, four kinds of database are developed. However some problems, such as ever-growing storage, inefficient access methods, and storage arrangement, arise from the implementation of a temporal database. Moreover, more users require a database which is easy to use, requiring less specialized knowledge and offering more output format than those of current packages.

This thesis consists of two parts: the development of a new efficient access method to a temporal database, and the implementation of a temporal database which combines graphic outputs, database functions, and an ability to handle temporal data efficiently.

A New Access Method of a Temporal Database

This new access method applies the concepts of a persistent search B-tree with path copying technique. Since the path copying technique copies nodes along a path from the root to the node, the persistent B-tree can keep

paths to each time version so that the old version can still be accessed. In chapter IV, a detailed performance analysis of the proposed method is provided. According to the analysis, the proposed method requires lower time complexity than those of approaches proposed by several authors [LUM84, BEN82, AHN86a, c]. However, the drawback of path copying technique is it requires spaces to keep new nodes which are copied along the path.

Implementation of a Temporal Water Resource Database

For the implementation of a temporal database, the indexing structure is enabled by two persistent B-trees, one for organizing time-stamp roots and the other for managing nodes which represent records of different time versions. Besides, a historical chain of an object can be provided by using pointers to connect tuples of different time versions. For physical organization, a sliding bar menu is designed and maintained to provide window facilities to help users to specify a query. Then, a graphical interface is designed and implemented to connect the DBMS and graphic system. Moreover, a graphical system is designed to store all the modules which implement the graphical display. Moreover, four graphic outputs are provided by the proposed database: geography, diagram, bar chart, line chart. Also, some utilities are available in this system, such as math functions and file reorganizer.

Contribution

Contributions of this thesis are discussed in four fields as follows.

Time Complexity

As mentioned before, the proposed method requires $O(\log_{m1} k + \log_{m} n)$ time complexity to retrieve an object in one time version and $O(\log_{m1} k + n \log_{m} n)$ to get all objects in one time version. These time complexities are lower than those needed by other approaches. Retrieving objects in each time version become more efficient with our proposed method.

Space Complexity

The proposed temporal database employs a persistent B-tree. Then, space for keeping embedded time attributes can be saved.

Convenience

Some conveniences provided by this proposed database are described as follows.

First, it provides users an easy way to specify a query. Since this temporal database employs a menu query method, even a non-professional user who does not know the structure of the database can specify a query easily.

Second, users can review facilities provided by this system quickly. As a user moves a cursor into a category,

a related window shows a list of functions. Therefore, users can know what facilities are available in each category.

Third, it provides more features than those of other database software. For instance, it provides a graphical display. Since the graphical display functions are provided by this system, a user can obtain the result of a query with a graphical format in addition to the traditional tabular format. The result of a trend analysis can be more meaningful to a user with graphical output.

Continuation

Since database schema are used to define the structure of a database, updating this database can be performed by changing the database schema. In addition, future possibility are these:

1. A geographical extension to other states;

Although the proposed database is designed for the state of Oklahoma, the same concepts can be applied to other states. By creating a window with all names of all states and modifying the database schema, the proposed database can manage water data for all states.

2. An extension to including application-users in addition to end-users.

In addition to using a menu query, a SQL-like language may be designed in future for technical users for a batch system.

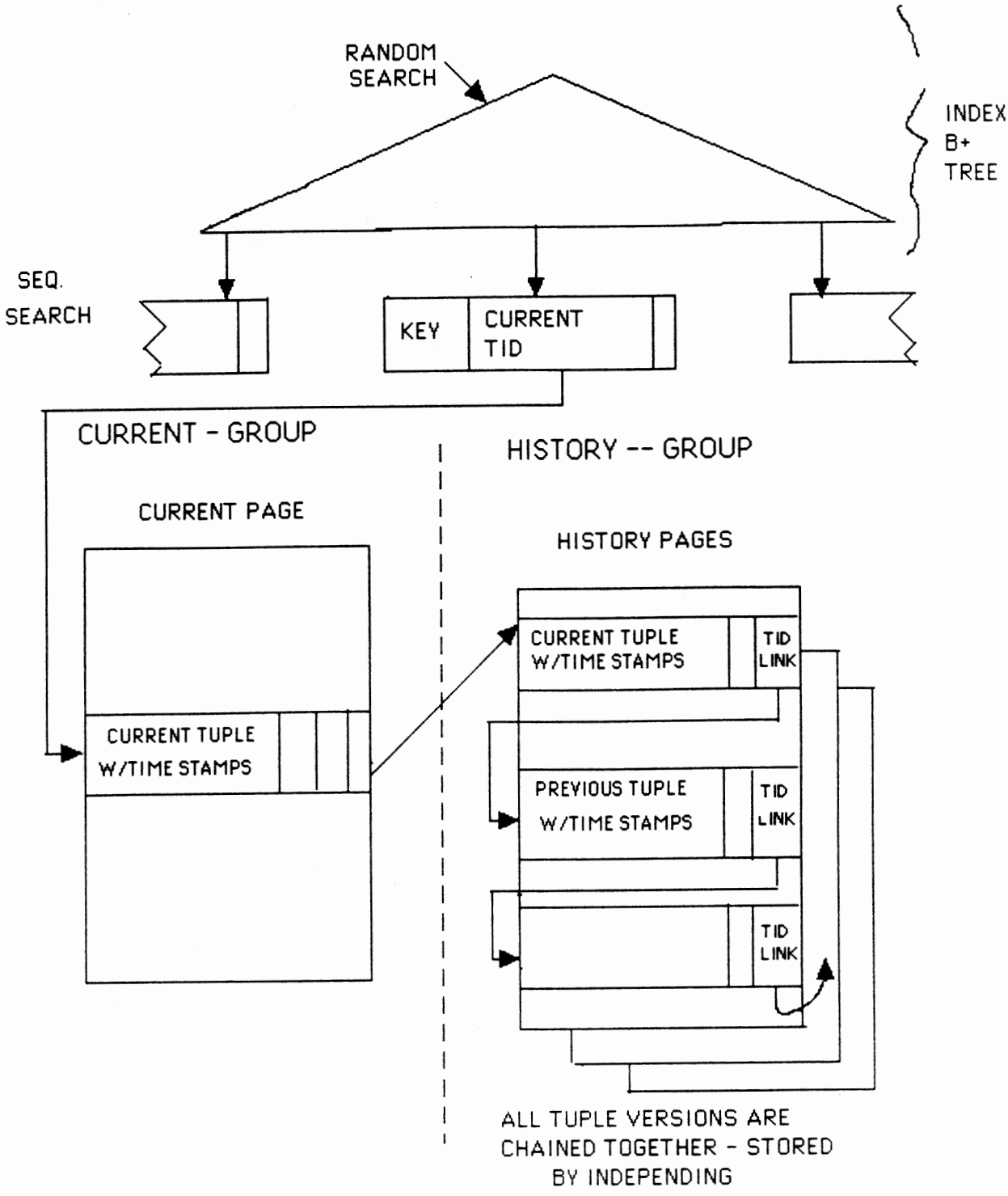


Figure 5. Ben-Zvi's Approach
(Source From [BEN82])

time 1: C, S, D
time 2: T, A, M
time 3: P, I, B

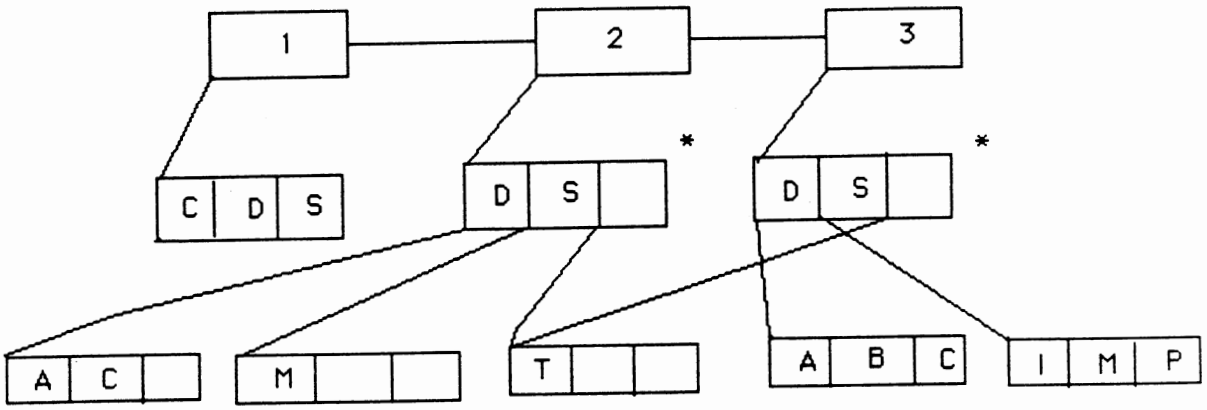


Figure 6. An Example of the Insertion Operation

time 1: C, S, D
time 2: T, A, M
time 3: (d) S, (i) P, I, B, W
time 4: (u) T

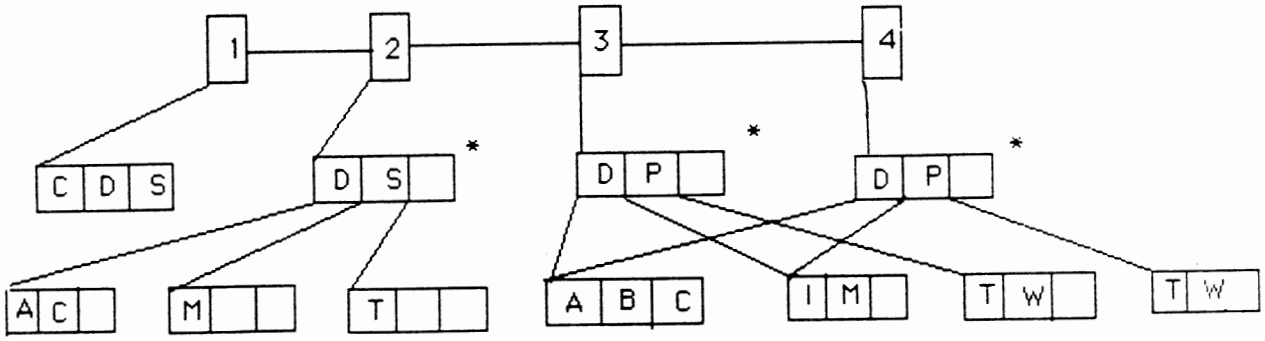


Figure 7. An Example of the Deletion and Update Operations

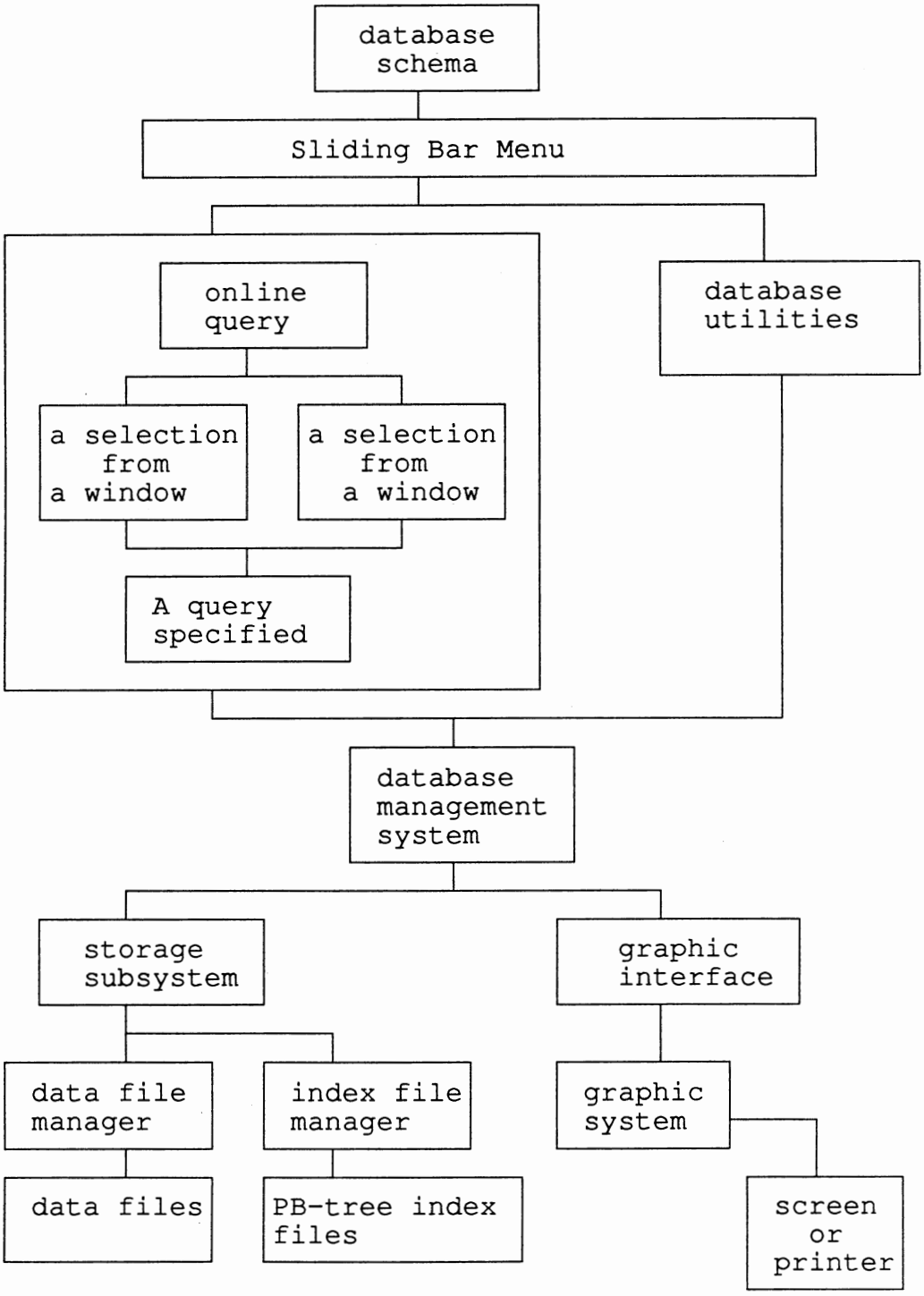


Figure 13. A Diagram of Overall Architecture of the Proposed Temporal Database

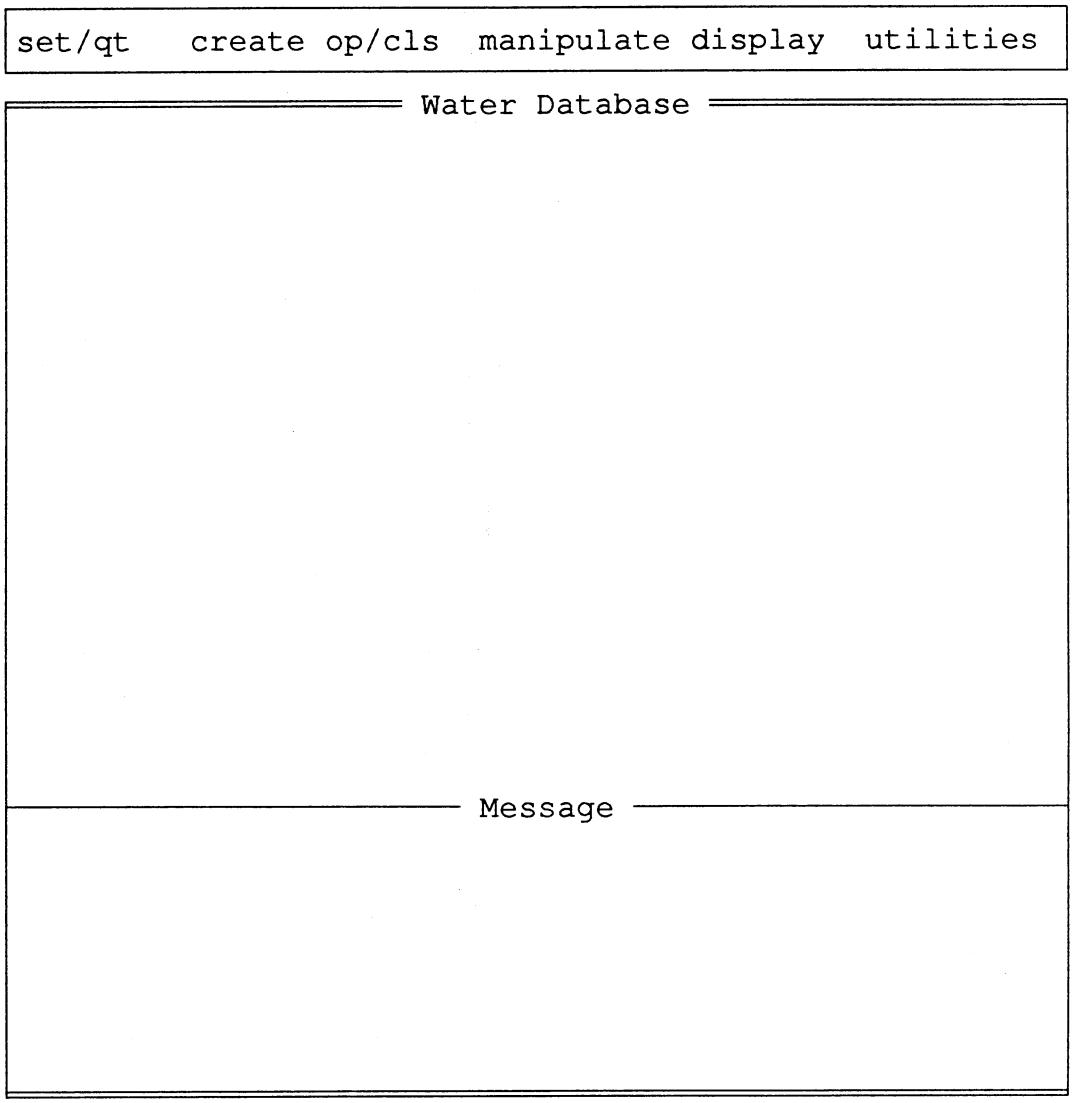


Figure 14. Categories of a Sliding Bar Menu

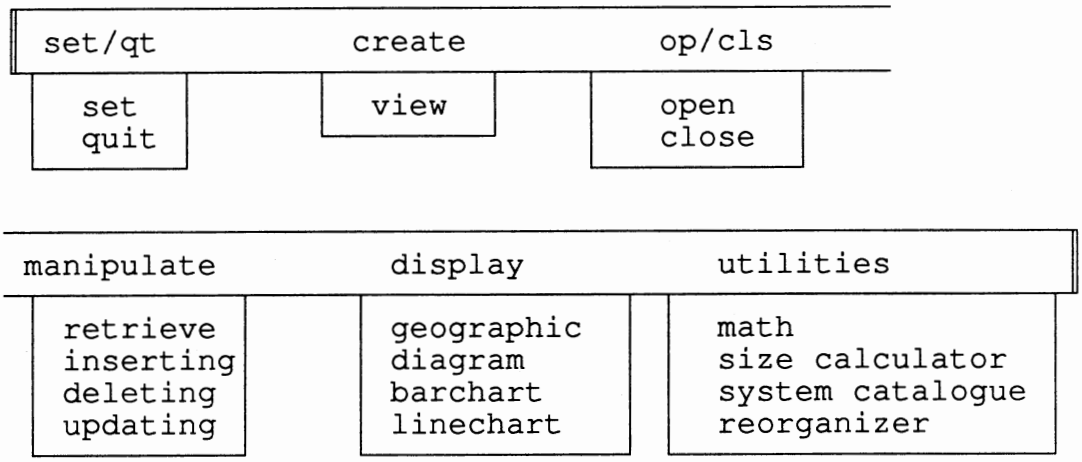
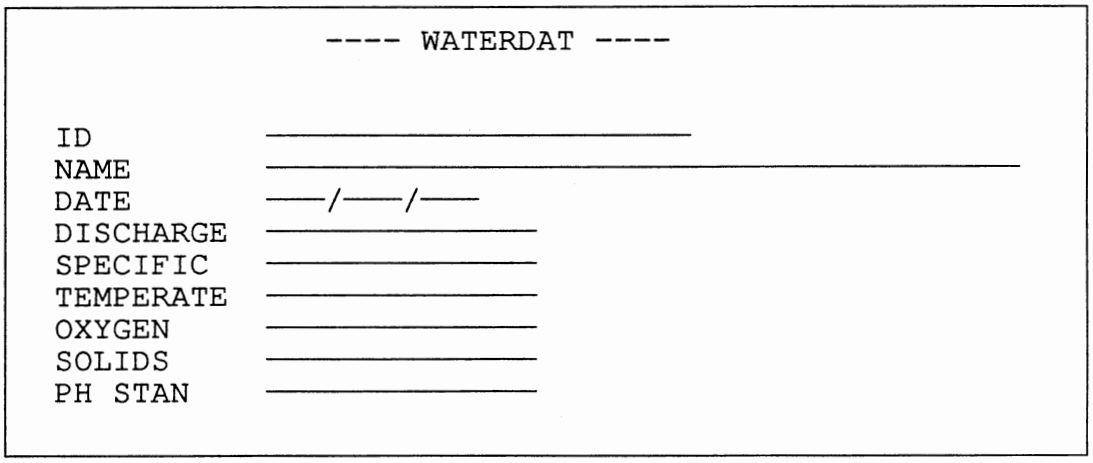
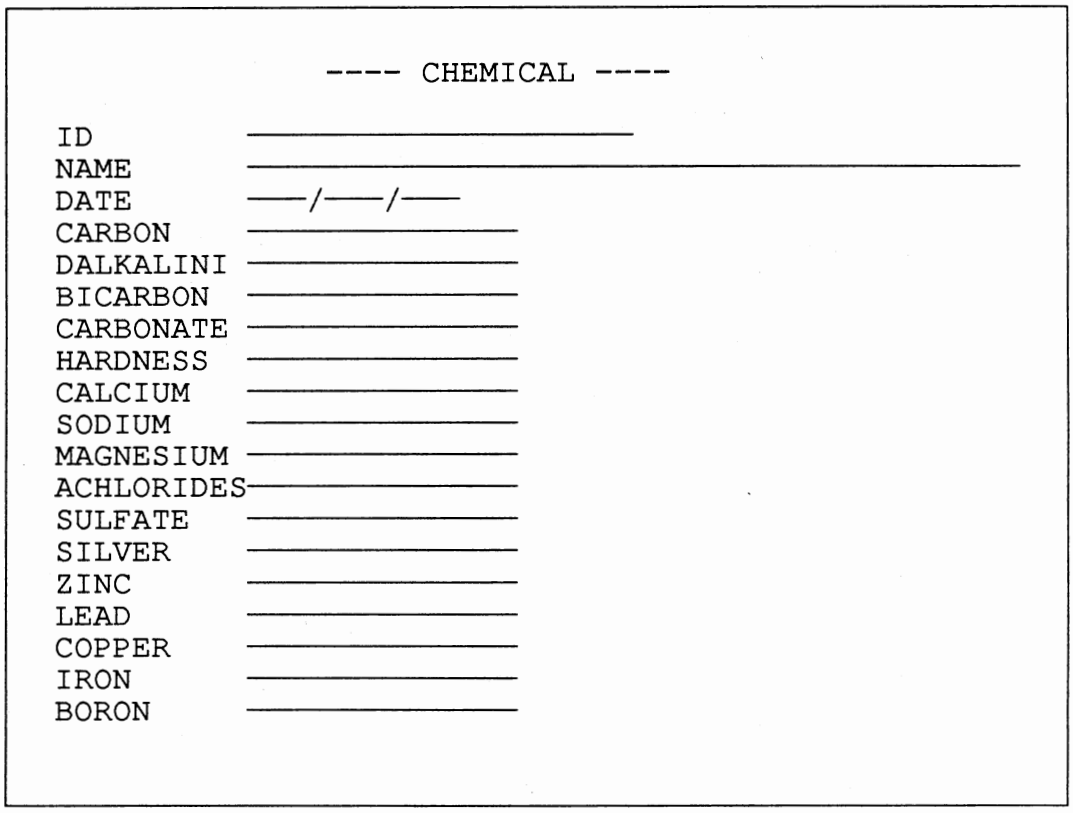


Figure 15. The Contents of Subwindows



a. The Input Templates for a WATERDAT Relation



b. The Input Templates for a CHEMICAL Relation

Figure 18. Formats for Input Templates

TABLE I

TIME COMPLEXITY OF APPROACHES
DISCUSSED IN CHAPTER V

	Ben	Lum
A	$O(k + \log_m n)$	$O(k + \log n2 + \log n))$
B	$O(n\ (k + \log_m n))$	$O(n\ (k + \log n2 + \log n)$

	Ahn	
	1	2
A	$O(k + \log_m n)$	$O(\log k + \log_m n)$
B	$O(n\ (k+\log_m n))$	$O(n\ (\log k + \log_m n))$

	Proposed Method
A	$O(\log_{m1} k + \log_m n)$
B	$O(\log_{m1} k + (n\log_m n))$

A: Retrieve One Object at a Time Stamp
B: Retrieve All objects at a Time Stamp
k: # of Time Stamps
m: The Order of B-tree
n: # of Node in The Tree
n2: # of Node in a History Index Tree
m1: The Order of a Time Stamp B-tree
1: Reverse Chaining in Ahn's Approach
2: Accession list in Ahn's Approach

TABLE II

THE SUMMARY OF COMPARISON
WITH OTHER DBMSs

	INGRES	DB2	dBaseIII Plus	WDIS	Proposed Database
1	Multi	Multi	Single	Multi	Single
2	Quel	SQL	a Menu and SQL-like Language		a Menu Query
3	None	None	None	None	Yes
4	None	None	None	None	Yes

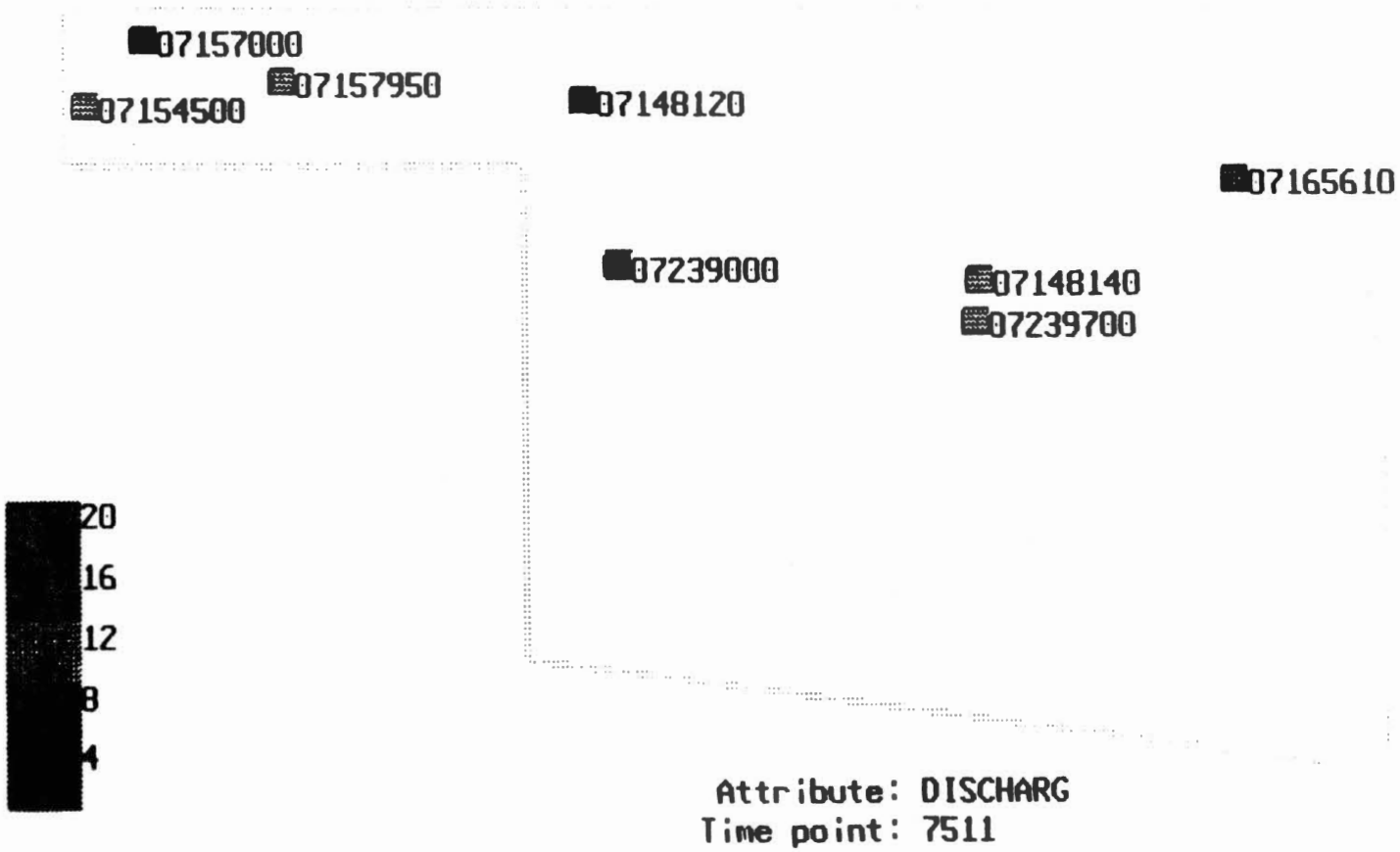
1: Users type
2: A Query Language
3: The Graphic Functions
4: A Special Ability to Handle Time

TABLE III

AN EXAMPLE OF A WATERDAT RELATION AT TIME POINT 75/11

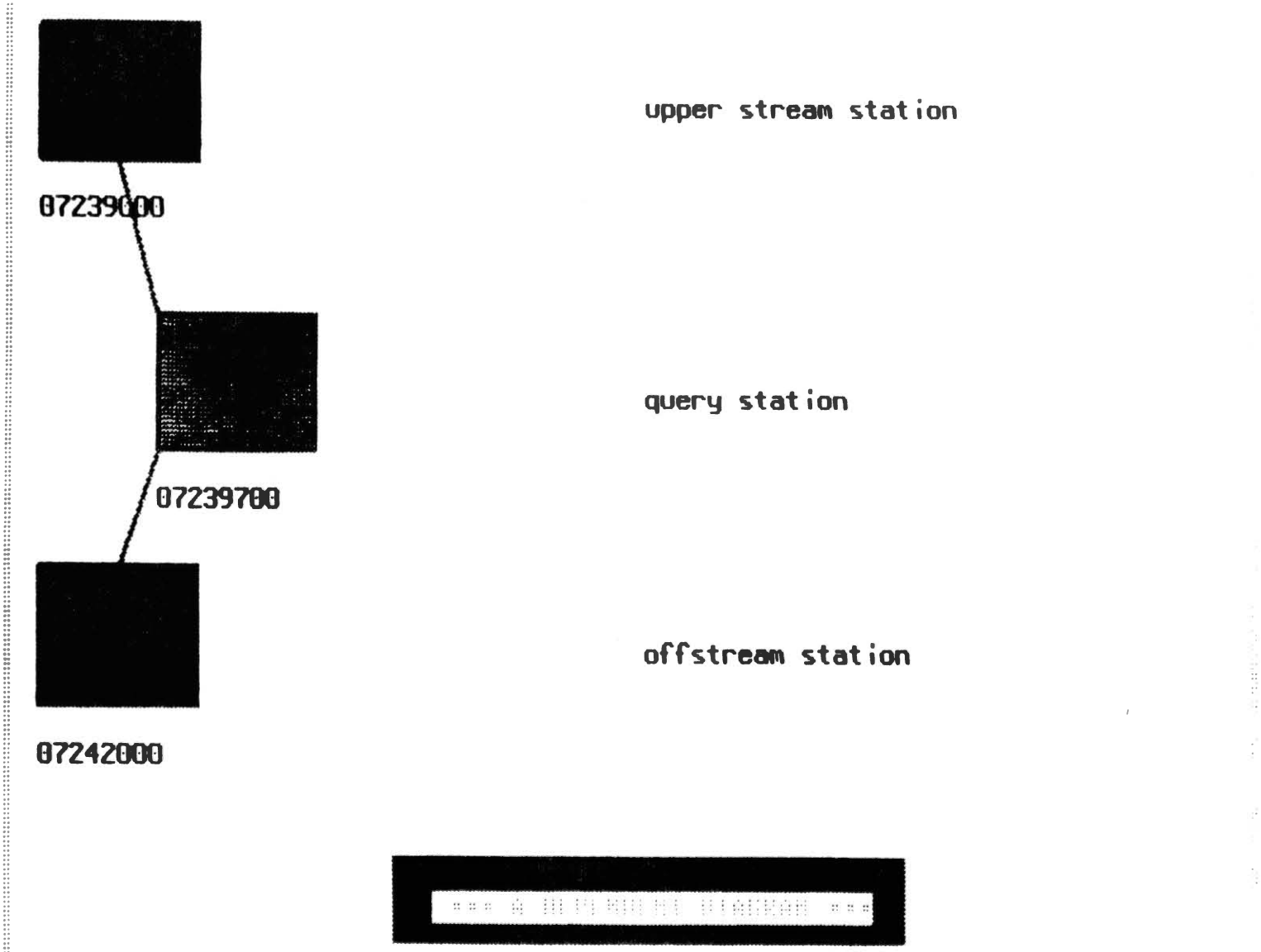
ID	NAME	DATE
07239000	NORTH CANADIAN RIVER AT CANTON, OK	751118
07242000	NORTH CANADIAN RIVER NR WETUNKA, OK	751105
07239700	NORTH CANADIAN RIVER NR YUKON, OK	751108
07148140	ARKANSAS RIVER, OK	751112
07148120	ARKANSAS RIVER, OK	751116
07165610	ARKANSAS RIVER AT MUKOGEE, OK	751121
07154500	CIMARRON RIVER NR KENTON, OK	751105
07157000	CIMARRON RIVER NR MOCANE, OK	751106
07157950	CIMARRON RIVER, OK	751123

DISCHARG	SPECIFIC	TEMPERATE	OXYGEN	SOLIDS	PH_STAN
19.0000	1500.0000		9.0000	12.0000	345.0000
21.0000	2100.0000	141.0000	11.0000	13.0000	300.0000
10.0000	1800.0000	223.0000	12.0000	21.0000	250.0000
13.0000	1300.0000	321.0000	10.0000	9.0000	230.0000
16.0000	1800.0000	145.0000	12.0000	13.0000	315.0000
17.0000	2500.0000	321.0000		8.0000	275.0000
11.0000	3000.0000	313.0000	16.0000	10.0000	
12.0000	2300.0000	212.0000	15.0000	6.1000	312.0000
11.0000	3200.0000	111.0000	13.0000	7.1000	210.0000



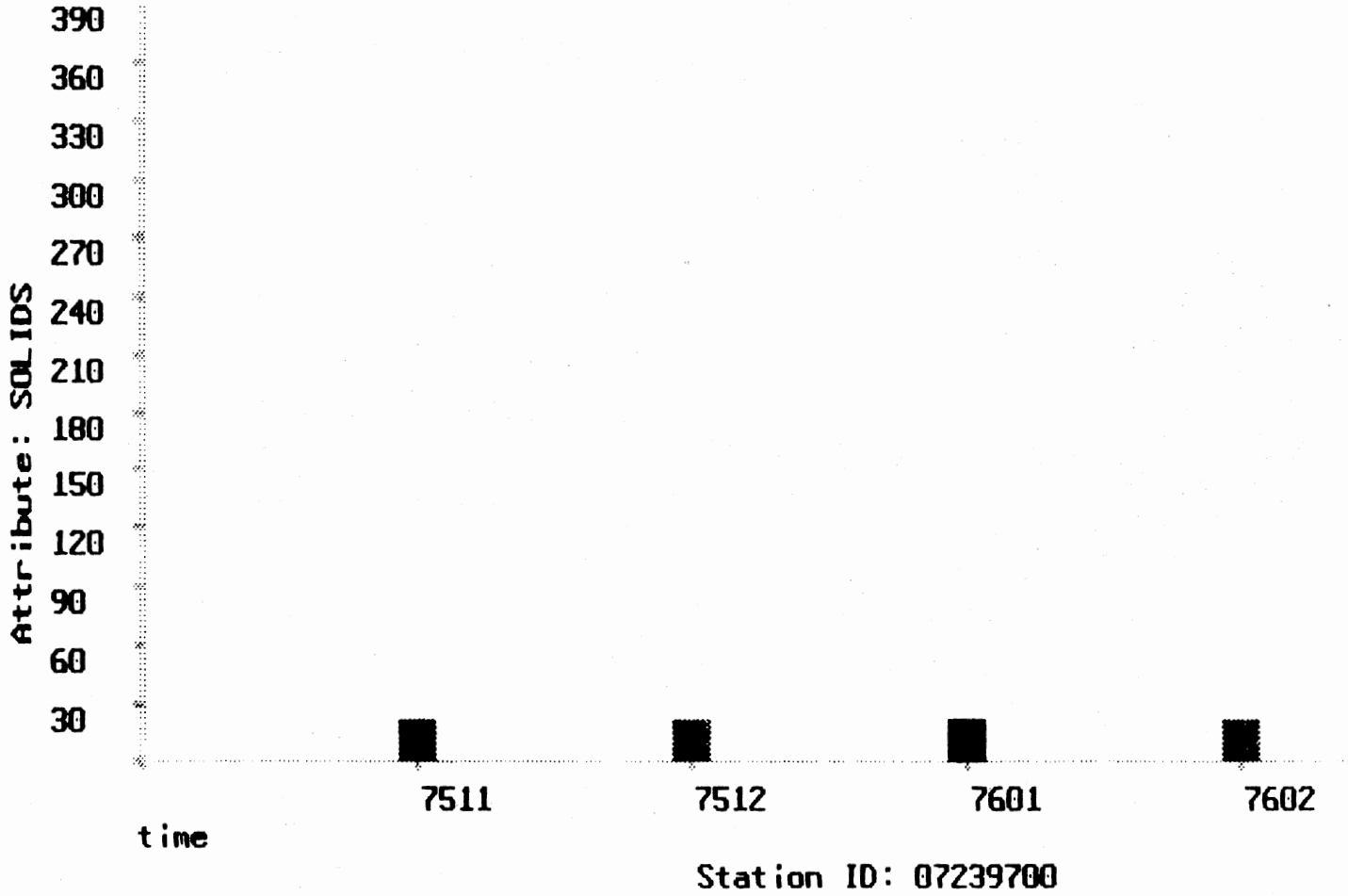
Unit: Milligrams per Liter (MG/L)
7511: November 1975

Figure 19. A Geographic Output Format



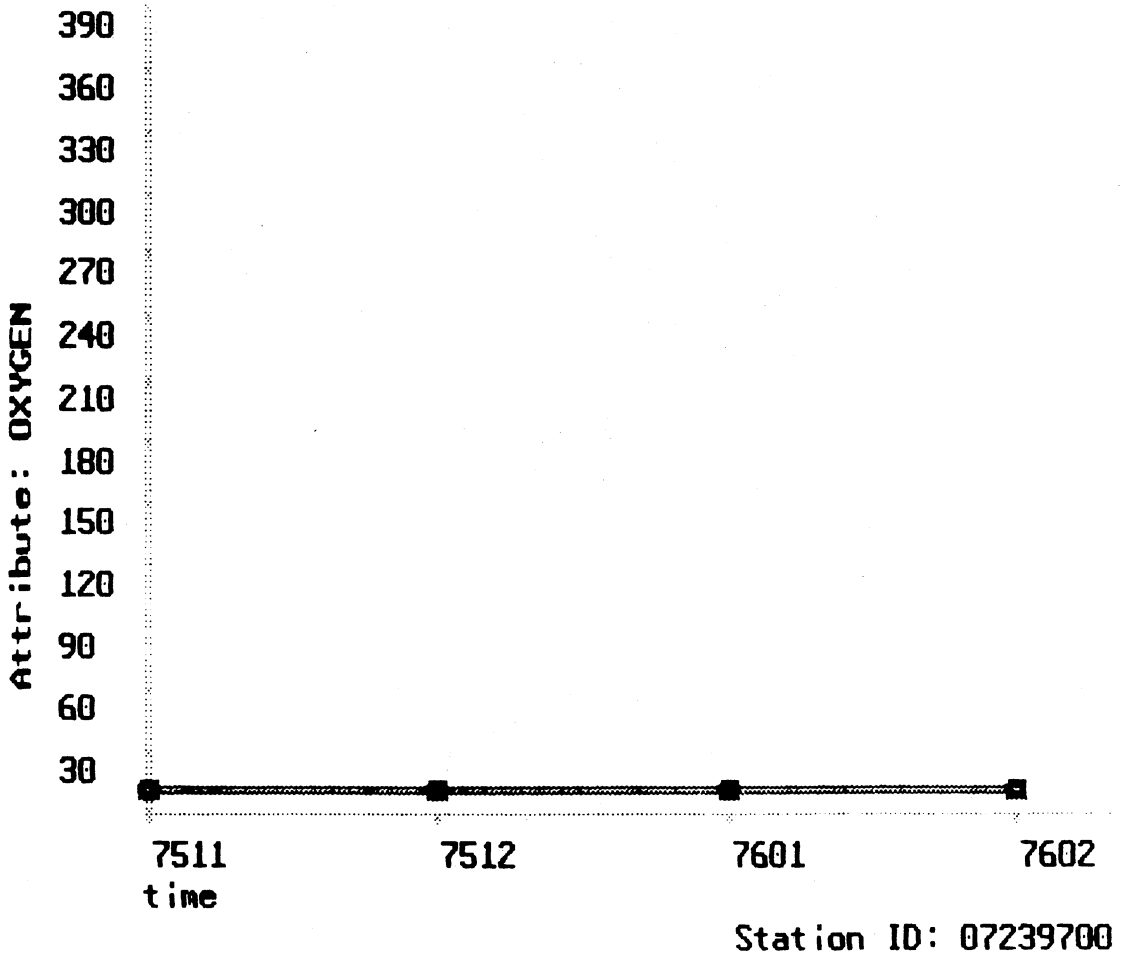
07239000: Station ID

Figure 20. A Dependent Output Format



Unit: (MG/L)
7511: November 1975

Figure 21. A Bar Chart Output Format



Unit: (MG/L)
7511: November 1975

Figure 22. A Line Chart Output Format

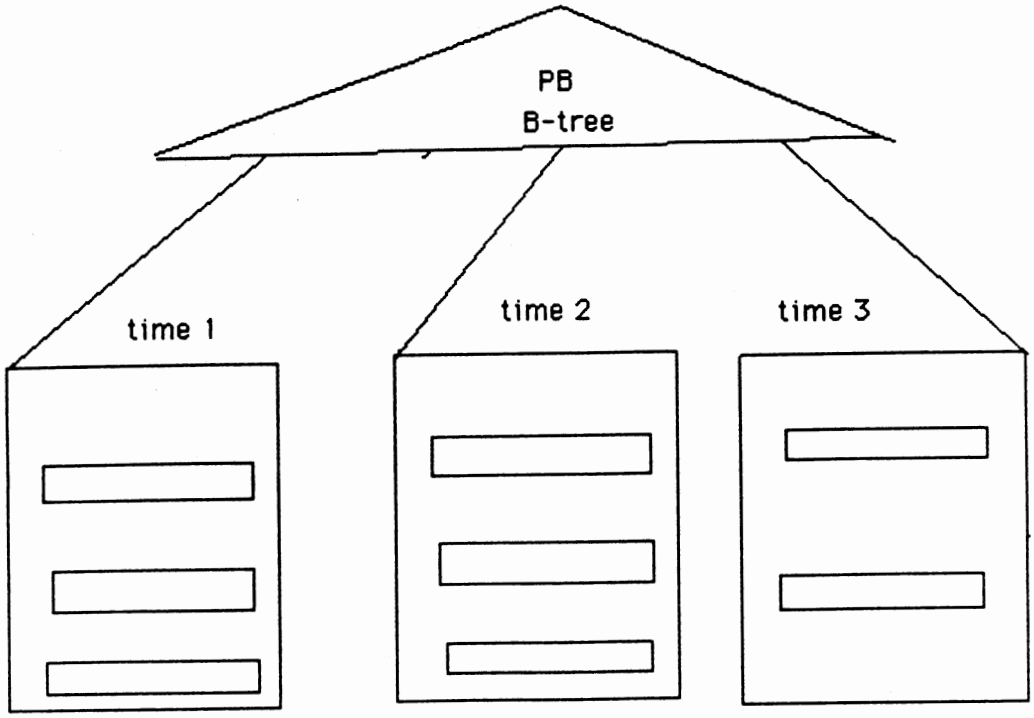


Figure 17. An Example of the Proposed Approach

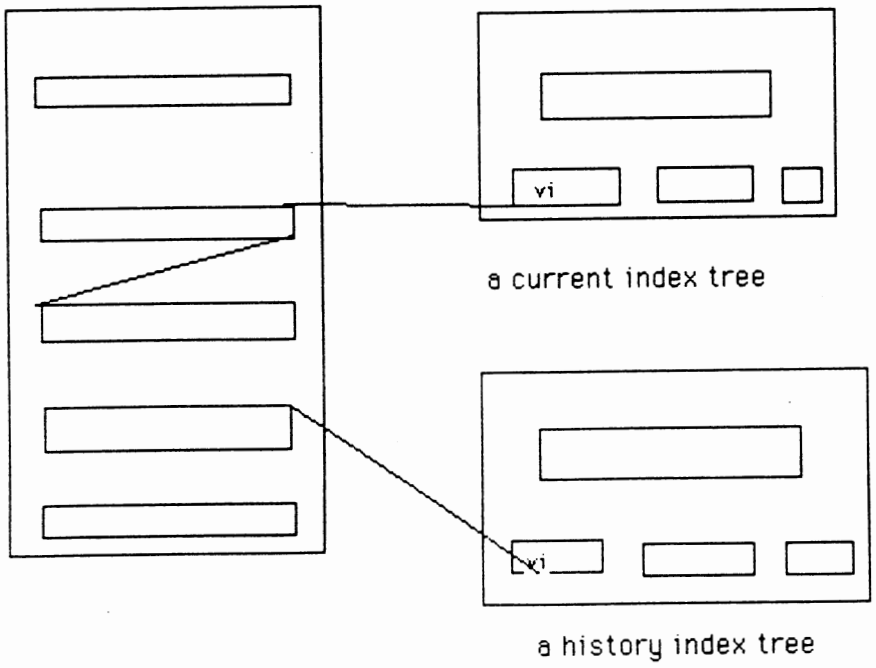
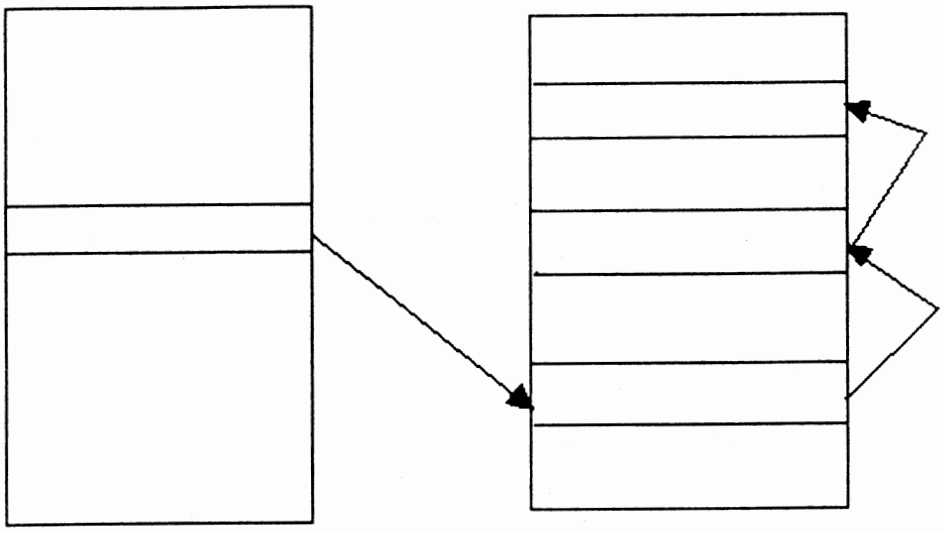
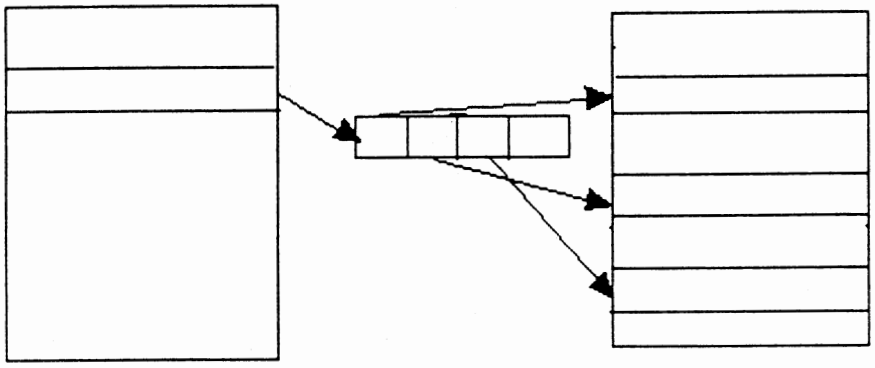


Figure 23. Lum's Approach (Source from [LUM84])



a. Reverse Chaining



b. Accession List

Figure 24. Ahn's approach
(Source from [AHN86a, c])

REFERENCES

[AHN86a] Ahn, I. "Performance Modeling and Access Method for Temporal Database Management Systems," Phd. diss. Computer Science Department, University of North Carolina at Chapel Hill, July 1986.

[AHN86b] Ahn, I. & Snodgrass, R. "Performance Evaluation of a Temporal Database Management System," in Proc. of ACM--SIGMOD International Conference on Management of Data, Ed. C. Zaniolo, Association for Computer Machinery, Washington, DC: May 1986, pp. 96-107.

[AHN86c] Ahn, I. "Towards an Implementation of Database Management Systems with Temporal Support," in Proc. of International Conference on Data Engineering, Feb. 1986, pp. 374-381.

[ANGE90] Angelaccio, M., Catarci, T. & Santucci, G. "QBD*: A Graphical Query Language with Recursion," IEEE Transactions on Software Engineering, Vol. 16, No. 10, Oct. 1990, pp. 1150-1163.

[ARIA86] Ariav, G. A. "A Temporally Oriented Data Model," ACM Transactions on Database Systems, Vol. 11, No. 4, Dec. 1986, pp. 499-527.

[BATO82] Batory, D. & Gottieb, C. "A Unifying Model of Physical Databases," ACM Transactions on Database Systems, Vol. 7, No. 4, Dec. 1982, pp. 509-539.

[BEN82] Ben-zvi, J. "The Time Relational Model," Phd. dissertation, UCLA, 1982.

[CHOW87] Chow, B. M. "Data bases: Getting the most for your Money, " Journal of the America Water Works Association, Vol. 79, No. 6, Jun. 1987, pp. 56-61.

[CLIF85] Clifford, J. & Tansel, A. U. " On an Algebra for Historical Relational Database: Two Views," in Proc. of ACM-SIGMOD Conference, 1985, pp. 247-265.

[COBU90] Coburn, E. J. "A Conceptual View of Temporal

Databases," IEEE Transaction on Software Engineering, 1990, pp. 170-172.

[COMP89] Computer Advance Research Committee, "Committee Report: Computers in the Water Industry," Journal of the America Water Works Association, Feb. 1989, pp. 74-79.

[FOGG84] Fogg, D. "Lessons from A "Living in a Database" Graphical Query Interface," ACM-SIGMOD, 1984, pp. 100-106.

[GADI85] Gadia, S. K. & Vaishnav, J. "A Query Language for a Homogeneous Temporal Database," in Proc. of Fourth Annual ACM-SIGMOD, 1985, pp. 66-83.

[GADI86] Gadia, S. K. "Toward a Multihomogeneous Model for a Temporal Database," in Proc. of the International Conference on Data Engineering, 1986, pp. 390-397.

[GADI88a] Gadia, S. K. "A Homogeneous Relational Model and Query Languages for Temporal Database," ACM Transactions on Database Systems, Vol. 13, No. 4, Dec. 1988.

[GADI88b] Gadia, S. K. & Yeung, C. S. " A Generalized Model for a Relational Temporal Database," in Proc. of ACM-SIGMOD Conference on Management Data, 1988, pp. 251-259.

[HANC87] Hancock, M. C. & Heaney, J. P. "Water Resources Analysis Using Electronic Spreadsheets," Journal of Water Resources Planning and Management, Vol. 113, No. 5, Sep. 1987, pp. 639-658.

[HERO80] Herot, C. F. "Spatial Management of Data," ACM Transactions on Database Systems, Vol. 5, No. 4, Dec. 1980, pp. 493-514.

[HYDR88b] Hydrodata User's Manual, USGS Daily and Peak Values, U S WEST Optical Publishing, U S WEST Systems, Inc., 1988.

[HYDR90a] Hydrodata QW User's Manual, Earthinfo, Inc., 1990.

[INDI90] Indira Singh & Ralf Beyer. "Water Resource Management System in Alberta Environment: An Empirical Analysis," in Proc. of the 7th Internal Engineering System Conference, 1990, pp. 618-625.

[JEFF85] Jeff R. Wright, ASCE A. M. & Kroll Peter. "

Rigid Format Database Management Using Micro Computer Technology," in Proc. of Computing in Civil Engineering, 1985, pp. 449-458.

[KAZE88] Kazerouni-Zand, M. and Fisher, D. D. "A Space efficient persistent B-tree," in the Proc. of Second Oklahoma Applied Computing Workshop, Tulsa, Oklahoma, March 1988, pp. 290-315.

[KAZE89] Kazerouni-Zand, M. and Fisher, D. D. "Deletion on Persistent B-tree," in the Proc. of third Oklahoma Applied Computing Workshop, 1989, pp. 90-96.

[KATZ84] Katz, R. H. & Lehman, T. J. "Database Support for Versions and Alternatives of Large Design Files," IEEE Transactions on Software Engineering, Vol. SE-10, No. 2, March 1984, pp. 191-200.

[KITT86] Kittridge, D. G. & Burkholder, W. F. "Water System Database for NAPLES, Florida, " Computing in Civil Engineering, Proc. of the Fourth Conference, Oct. 1986, pp. 27-31.

[LUM84] Lum, V., Dadam, P., Erbe,R. & Guenauer, J. " Designing DBMS Support for The Temporal Dimension ," in Proc. of the ACM-SIGMOD Conference, June 1984, pp. 115-130.

[MCKE87] Mckenzie, E. & Snodgrass, R. "Extending the Relational Algebra to Support Transaction Time," in Proc. of ACM-SIGMOD International Conference on Management of Data, 1987, pp. 467-478.

[MAIM89] Mainmone, Mark, "Developing a database for use in Groundwater Management," Journal of Water Resources Planning and Management, Vol. 115, No. 1, Jan. 1989, pp. 75-93.

[MERV90] "California Water Database," PUBLIC WORKS, Feb. 1990, pp. 62-63.

[OVER82] Overmyer, R. & Stonebraker, M. "Implementation of a Time Expert in a Data Base System," SIGMOD Record, Vol. 12, No. 13, Apr. 1982, pp. 51-59.

[ROTE87] Rotem, D. & Segev, A. "Physical Organization of Temporal Data," in Proc. of the Third IEEE International Conference on Data Management, 1987, pp. 547-553.

[SARN86] Sarnak, N. & Tarjan, K. E. "Planar Point Location Using Persistent Search Tree," Communication of the ACM, Vol. 29, No. 7, July 1986, pp. 669-679.

[SEGE87] Segev, A. & Shoshani, A. "Logical Modeling of Temporal Data," in Proc. of ACM-SIGMOD Conference, 1987, pp. 454-466.

[SHOS86] Shoshani, A. & Kawagoe, K. "Temporal Data Management," in Proc. of VLDB, 1986, pp. 79-88.

[SNOD85] Snodgrass, R. & Ahn, I. "A Taxonomy of Time in Database," in Proc of ACM-SIGMOD International Conference on Management of Data, May 1985, pp. 236-246.

[SNOD86a] Snodgrass, R. & Ahn, I. "Temporal Database," IEEE Computer, Vol. 19, No. 9, Sep. 1986, pp. 35-42.

[SNOD86b] Snodgrass, R. "A Temporal Query Language," ACM Transactions on Database Systems, Sep. 1986, pp. 35-42.

[SNOD87] Snodgrass, R. "The Temporal Query Language TQuel," ACM Transactions on Database Systems, Vol. 12, No. 2, June 1987, pp. 247-298.

[SPOO84] Spooner, D. L. "Database Support for Interactive Computer Graphics," ACM-SIGMOD, 1984, pp.90-99.

[STEV87a] Stevens, A. Turbo C Memory-Resident Utilities, Screen I/O and Programming Techniques. Oregon: Management Information Source, Inc., 1987.

[STEV87b] Stevens, A. C Database Development. Oregon: Management Information Source, Inc., 1987.

[STEV89] Stevens, R. T. Graphical Programming in C. CA: M & T Publishing, Inc., 1989.

[TANS89] Tansel, A. U., Arkun, M. E. & Ozsoyoglu, G. "Time-by-Example Query Language for Historical Databases," IEEE Transactions on Software Engineering, Vol. 15, No. 4, Apr. 1989, pp. 464-478.

[TANS86] Tansel, A. U. "Adding Time Dimension to Relational Model and Extending Relational Algebra," Information System, Vol. 13, No. 4, 1986, pp. 343-355.

[VERM87] Verma, V. & Lu, H. "A New Approach to Version Management for Database," National Computer Conference, 1987, pp. 645-651.

[WATS81] "WATSOTRE: A WATer STOrage and REtrieval system," U. S. Department of Interior/ Geological Survey, 1981.

VITA

Yun-chen Dunn

Candidate for the Degree of

Master of Science

Thesis: A NEW ACCESS METHOD AND IMPLEMENTATION OF A TEMPORAL WATER RESOURCE DATABASE

Major Field: Computer Science

Biographical:

Personal Data: Born in Taiwan, Republic of China, January 10, 1964, the daughter of Kan Shieh Teng and Tai Mei Soong.

Education: Graduated from Gi Mei Girls' High School, Taipei, Taiwan, Republic of China, in June, 1981; received Bachelor of Science Degree in Business Administration from National Chengchi University at Taipei in June, 1986; completed requirements for the Master of Science degree at Oklahoma State University in July, 1991.

Professional Experience: System Assistant, ST. Products Corp., Taipei, June, 1986, to July, 1988.