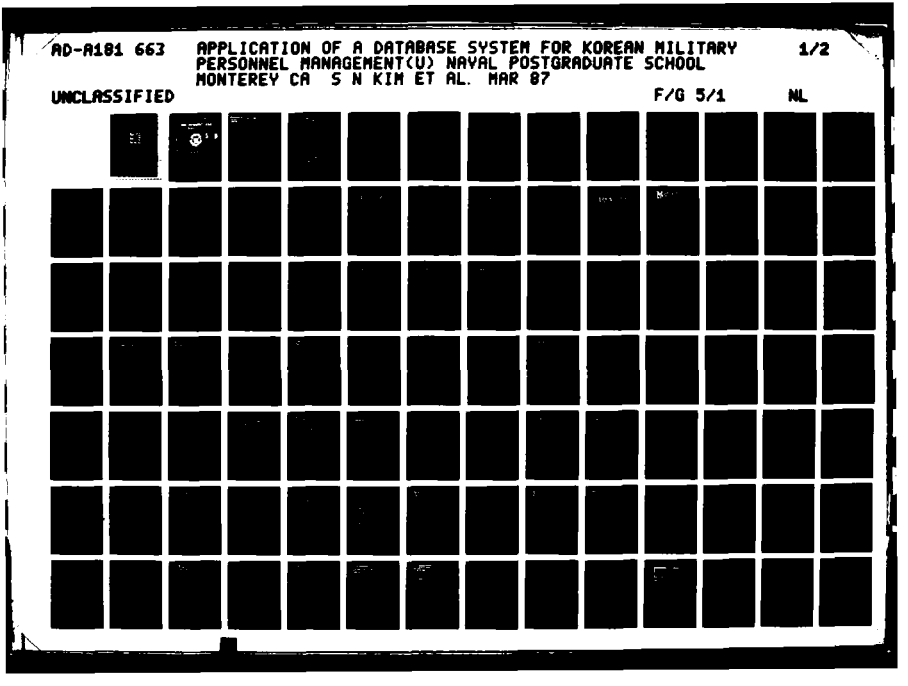
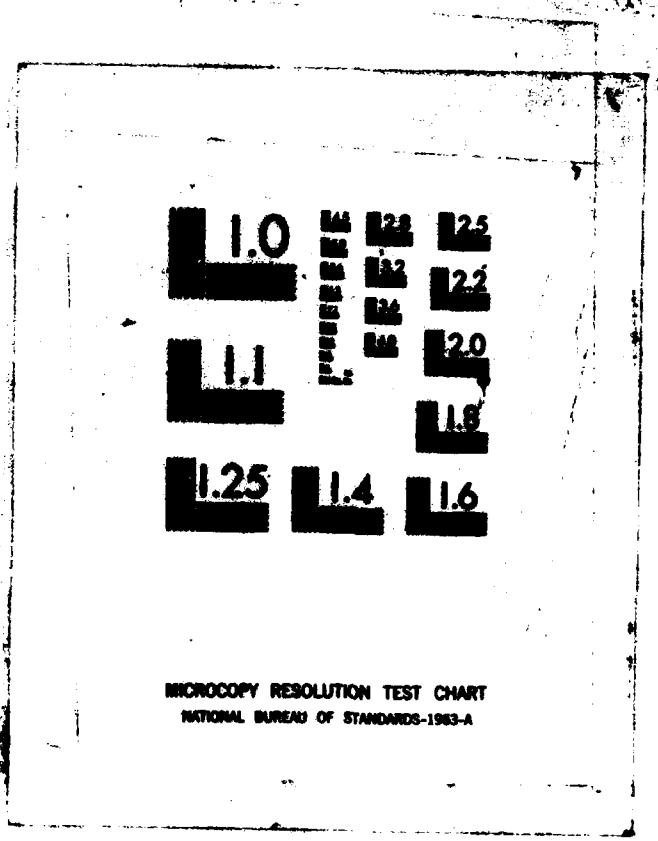
AD-A181 663 APPLICATION OF A DATABASE SYSTEM FOR KOREAN MILITARY PERSONNEL MANAGEMENT(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA S N KIM ET AL. MAR 87 1/2

UNCLASSIFIED F/G 5/1 NL




1.0
1.1
1.25
1.4
1.6
2.0
2.2
2.5
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

APPLICATION OF A DATABASE SYSTEM FOR KOREAN MILITARY PERSONNEL MANAGEMENT

by

Kim, Sam Nam
and
Park, Jae Bock

March 1987

Thesis Advisor Norman R. Lyons

Approved for public release; distribution is unlimited.

unclassified

SECURITY CLASSIFICATION OF THIS PAGE

A181663

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; Distribution is unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) 54	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)

APPLICATION OF A DATABASE SYSTEM FOR KOREAN MILITARY PERSONNEL MANAGEMENT

12 PERSONAL AUTHOR(S)

Kim, Sam N. and Park, Jae B.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1987 March	177

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Database system, Relational model, normal form, database design, data dictionary, dBASE III+ implementation, personnel management

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis present a application of data base system for Republic of Korean military personnel management system. In order to maximize the utilization of personnel resources computerized personnel information system is needed.

An important consideration in database design is to assure that it can be used for a wide variety of application and can be changed quickly and independently. For this purpose, we discuss about normal forms including functional dependency concept.

A simple data base using dBASE III+ is implemented with IBM pc, and is designed for the user who does not have computer experience, and is based on the thearetical design problems. (These).

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Prof. Norman R. Lyons	22b TELEPHONE (Include Area Code) 408-373-6685 / 22c OFFICE SYMBOL 54Ly

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Approved for public release; distribution is unlimited.

Application of a Database System for Korean Military Personnel Management

by

Kim, Sam Nam
Major, Republic of Korea Army
B.S., Republic of Korea Military Academy,1977

and

Park, Jae Bock
Captain, Republic of Korea Air Force
B.S., Republic of Korea Air Force Academy,1981

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
March 1987

Authors: _____
Kim, Sam Nam

_____
Park, Jae Bock

Approved by: _____
Norman R. Lyons, Thesis Advisor

_____
Richard A. McGonigal, Second Reader

_____
W. R. Greer, Jr., Chairman,
Department of Administrative Science

_____
Kneale T. Marshall,
Dean of Information and Policy Sciences

ABSTRACT

This thesis presents a database application, the Republic of Korea military personnel management system. In order to maximize the utilization of personnel resources a computerized personnel information system is needed.

An important consideration in database design is to assure that data can be used for a wide variety of applications and can be changed quickly and independently. For this purpose, we discuss normal forms including functional dependency concepts.

A simple data base using dBASE III+ is implemented on an IBM PC. It is designed for the user who does not have computer experience, and is based on the theoretical design problems.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENT

We would like to give our deepest appreciation to our country, the Republic of Korea, and particularly the Korean army and Air Force for investing the time and money sponsoring us in this graduate program.

Furthermore, we want to give our sincere thanks and appreciation to our thesis advisor, Professor Norman R. Lyons, whose long hours of consultation with us and willing support aided inmeasureably in the completion of this thesis.

In addition, our second reader, Professor Richard A. McGonigal, also was instrumental in helping complete this project with significant comments and useful suggestions.

We want to also thank Captain Steven Sudderth and Lt/commander John Haima who provided consultation concerning English translation.

Finally, we want to give a special thanks to our wives, Myung Ok and Eun Kyung for their gracious understanding and patience throughout our graduate study and particularly with the long hours spent in the completion of this thesis.

I. INTRODUCTION

The influence of personnel management on the business organization has increased appreciably in recent years. Much of this increase can be attributed to the growing complexity of human resource management and the issues related to it. In the Republic of Korea(R.O.K), in order to strengthen the capability of combat under the limited national defense expenditure, it is imperative that personnel management be performed very efficiently. To achieve this goal, the high level managers of the military very often need a variety of data relevant to each person. Therefore, a modern database management system is needed for the R.O.K military personnel management system.

The database management systems now in use permit greater flexibility in meeting information requirements, faster response times, and easier user access to stored data then earlier software systems. These benefits are achieved at the expense of larger capital and manpower investments, greater system complexity, lower processing efficiency, and long pay off periods. The value of such systems can not be determined strictly on money, but also by the increase in the number of applications processed for noncomputer oriented users. Perhaps the most exciting development to occur from the introduction of such systems is the wide availability of easy to use query type languages which permit nonprogrammers to create, update, maintain, and extract information from their own files.

In database development, it should be possible to query the database to satisfy the user's requirements using application programs or the Database Management System(DBMS) itself. Because there are many types of data structures, models, designs etc., we should select one which depends on the problem or situation. The normal form concepts of relational database models will be applied to develop a database for the Korean military personnel management. Most experts agree that the relational data model supports data independence better than other models.

This thesis will focus on a database design applying a Relational Model to R.O.K Army and Air Force personnel management. The actual sample data of R.O.K Army personnel will be used for implementing the sample program using dBASE III+. In Chapter II, Background, we will address personnel management and give a general

overview of a database system. In Chapter III, general concept of the Relational Model will be discussed and in Chapter IV, data base design problems will be discussed in detail theoretically. In Chapter V, practical system analysis and relational database design for Korean Army personnel management system will be discussed. In Chapter VI, we will discuss and show example for data base implementation and in Chapter VII, processing procedure to access the sample program in Appendix A will be shown. In Chapter VIII, conclusions will be drawn and recommendations for implementation will be offered.

II. BACKGROUND

A. PERSONNEL MANAGEMENT

1. Basic concept

To meet organizational objectives it is necessary to continually acquire human resources; integrate employees into the organization; develop employee potential; and maintain the work force. Personnel management is an integral part of the broader field of management. Management has been defined as the process of accomplishing objectives through the efforts of other people within an organization. Thus, we can say that the role of personnel is critical for managing in general. [Ref. 1: p.17]

In any military hierarchy there are many levels of command and control. Each level has its own duties and needs adequate number of people with respect to knowledge, capability, rank, skill etc.

2. Objectives.

Objectives are the starting point of the management process. They give the organization and its people a purpose and direction. Objectives serve to guide managers and employees in their efforts. The managers commonly perform these functions:

(1) Planning - determining strategies and programs to help accomplish established objectives.

(2) Organizing - grouping and assigning activities, staffing the organization, and delegating authority to carry out activities.

(3) Directing - encouraging human efforts and stimulating accomplishment of objectives.

(4) Controlling - measuring accomplishments, comparing results with planned objectives, determining causes of deviations, and taking necessary corrective action.

3. Korean Army and Air Force personnel management

The Republic of Korea's(R.O.K) Army and Air Force uses the general staff model which includes Personnel, Intelligence, Operations and Training, and Logistics(G1-G4). The R.O.K government spends a large percentage of the total government budget for national defense and the Department of National Defense spends a significant portion of the national defense expenditures for personnel.

Personnel managers need data about the individual personnel capability and group personnel capability to analyze, to investigate, to plan, and to apply this data for

their organizations. Information about group personnel power can be derived by a collection of individual personnel power. It is important to increase individual and group personnel power in the personnel management field so that the right people move into the right jobs at the right times and under the right circumstances. [Ref. 2: p.73]

In the military, information about individual personnel power can be derived from functions involving procurement, education and training, assignment, treatment, promotion and retirement. In order to reduce the national defense expenditure and increase the war_making capability, the Korean military needs a computerized management information system personnel management. Therefore, some important functions of the Army and Air Force's Department of Personnel Management and other essential information are analyzed as system requirements. Information management by computer is very important for fast and accurate personnel management in the Korean military.

B. DATA VERSUS INFORMATION

Data and information are meant to have two distinct meanings. Data may best be thought of as representing objective, external realities such as a flash of lightning in the sky, the expression on an employee's face, or the number of widgets produced per day on the production line. Viewed in this way, data becomes pure fact. Data is knowledge for the sake of knowledge. When captured and stored, data is merely a record of these specific characteristics and events which can be reliably observed and which have sufficient impact to be taken note of.

On the other hand, the term "information" may be restricted to mean interpreted data. Information should be thought of as the statement of the relationship of any given characteristic or event to specific goals and purposes. It is what is used to control progress toward these goals and objectives. The term "information" will be reserved to mean knowledge for the sake of purposeful action. These are the key definitions:

> Data is the record of any reliably observable characteristic or event in nature. Information is the description of the relationship of any such characteristic or event to human goals and/or business purposes [Ref. 3: p.124].

Figure 2.1 shows this relationship.

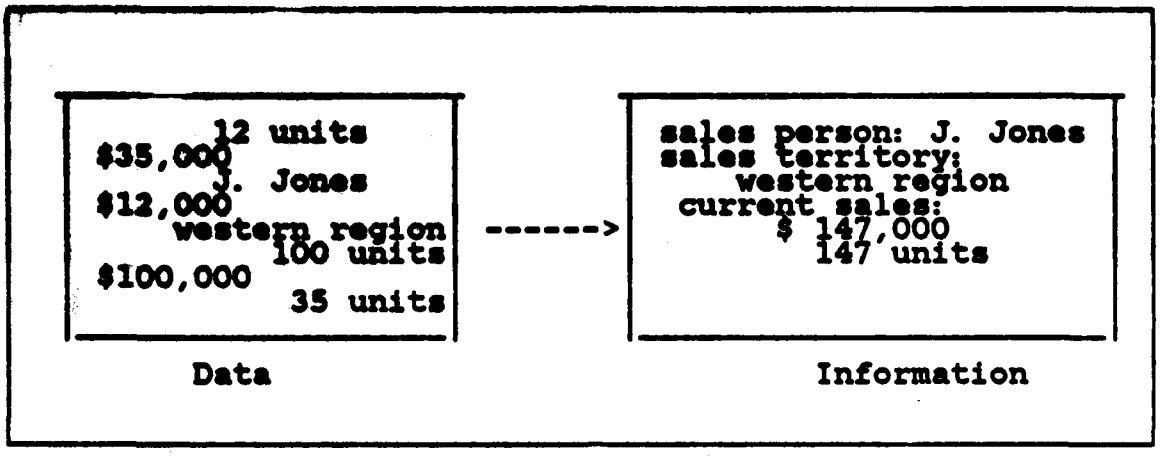


Figure 2.1 Data versus Information.

C. GENERAL OVERVIEW OF A DATABASE SYSTEM

1. Introduction

The theory of data management predates computers. Early attempts at putting the theory into practice with rudimentary equipment were made in the 1940s and early 1950s. Computers were applied to the management of data in late 1950s and early 1960s. These computers were able to process data more quickly and in greater quantities than ever before, but the management of data (storage, manipulation and retrieval) was still quite unsophisticated. The architecture of computers at that time facilitated sequential processing of large volumes of data or massive computations made on small amount of data, one job at a time. In the middle 1960s, computer architecture was radically changed. A quantum increase in the size of computer memory and the introduction of operating systems made it possible for computers to do more than one job concurrently. This kind of processing called multi-programming, has continued right up to the present.

Concurrent with multi-programming came the capacity to do what is called "on-line" or single transaction processing. Rather than process large volumes of data sequentially, it has become economically feasible to access specific information from computer stored files within seconds. In the late 1960s, more sophisticated methods of storing and retrieving data were incorporated into computer software (programs). These programs were the first data base management systems. The idea was just a little ahead of its time. Although computer memories had grown in size from thousands of positions to hundreds of thousands, they were not quite up to the task.

However, in the early and middle 1970s, computer memory capacities were such that millions of characters could be stored in them, and storage technology had increased the potential size even further. This increase has made possible the implementation of data management software. Since it has become technologically possible (and is at least approaching economic feasibility), the concept of data management is now emerging in the business community. Technological advances are making it possible to store data in a way that is radically different from most of the contemporary methods now in use. This new technology is manifesting itself in both hardware and software. Hardware technology is allowing for large amounts of data (billions of characters) to be stored on-line. Software technology is supplying the mechanisms for the storing, updating and retrieving of that data.

The mechanisms for manipulating and retrieving data (converting data to information) are known as Data Base Management Systems(DBMS). There are a number of software packages that provide these mechanisms. [Ref. 4: p.11]

2. **Data base system definitions**

Data base terminology and data base theory will be briefly presented in this section. To facilitate this discussion, it is necessary to set some common definitions. They are:

(1) Data is a group of non-random symbols that represent quantities, actions, things, facts, concepts or instructions in a way suitable for communication and processing by humans or machines. Information is data that has been processed and presented, as described in B. in this chapter.

(2) A record is a group of related data items.

(3) A file (data set) is a collection of related records. A database is a collection of files logically related in such a way as to improve access to the data and minimize redundancy of data.

(4) A data base management system is a set of programs that function to create and update the data base, retrieve data and generate reports from the data base. A conceptual model (data model) is a representation of the information content of the data base. Figure 2.2 shows the composition of data base.

3. **What is DBMS?**

In simple terms, it is a computer software system that provides control, retrieval and storage of data contained in one or a combination of data files that are tied together by the DBMS and are more commonly referred to as a data base. Provided by the computer manufacturer or an independent software house, the software package is adaptable to all application systems. But DBMS can be truly understood by contrasting it with traditional practices.

The ordinary systems development approach has been to organize data into

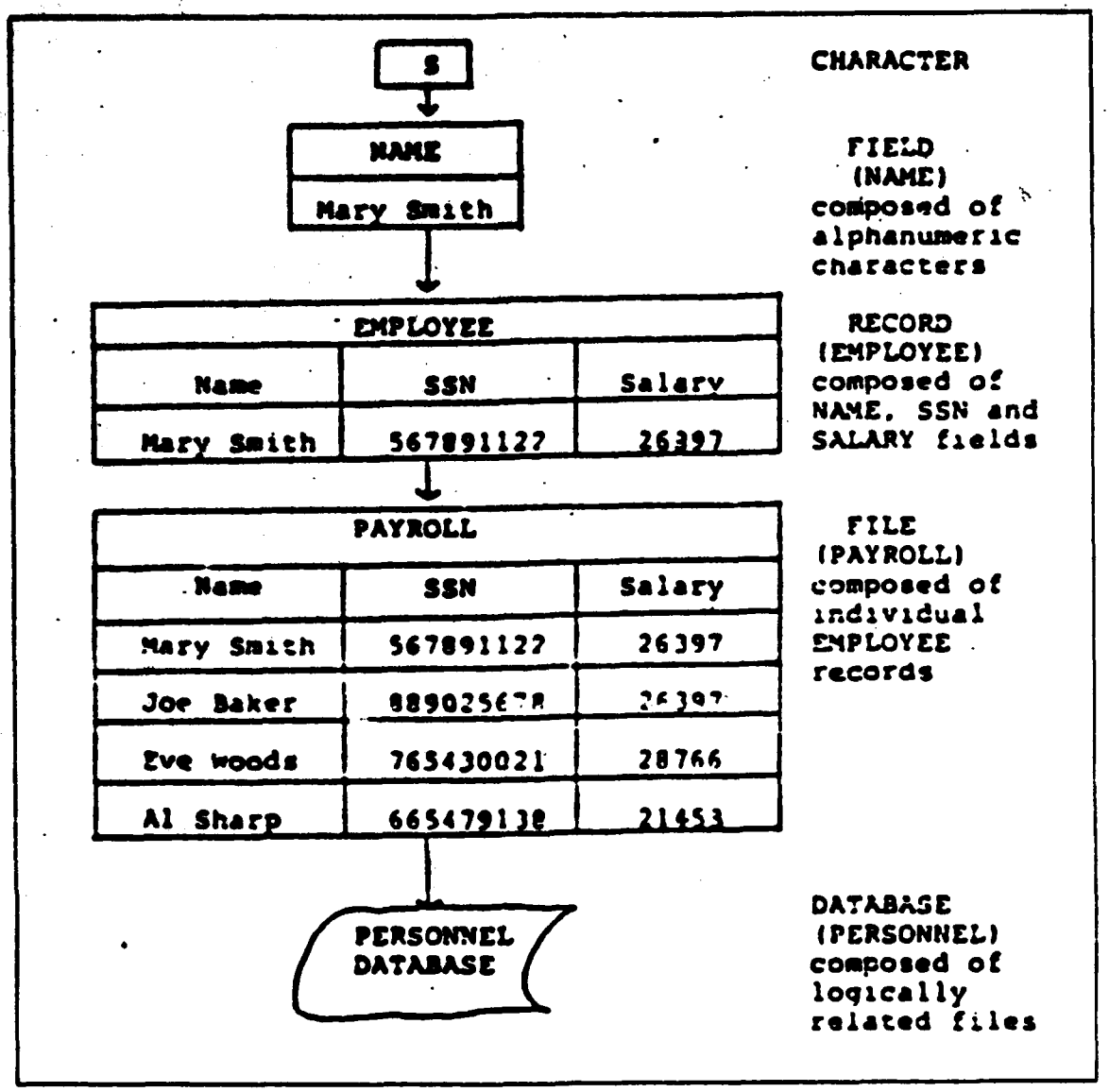


Figure 2.2 Composition of database[1].

individual files, with the typical business data processing center developing around the computerization of separate applications such as payroll, inventory, and accounts receivable. What happens, however, is that the computer data files necessary to support these applications are fixed in their structure with preset formats frozen into the computer programs. The result has been that when a range of data values has grown to the point where, let us say, one more digit position is required, there may be

[1]Roberstone, Debra L., *Data Dictionary Systems and their role in information resource management*, Naval Postgraduate School, Mar 1984

no available physical space in the file record. The entire computerized data record must then either be expanded or redesigned, and this new design in turn leads to related changes in all computer programs that use these computer data files.

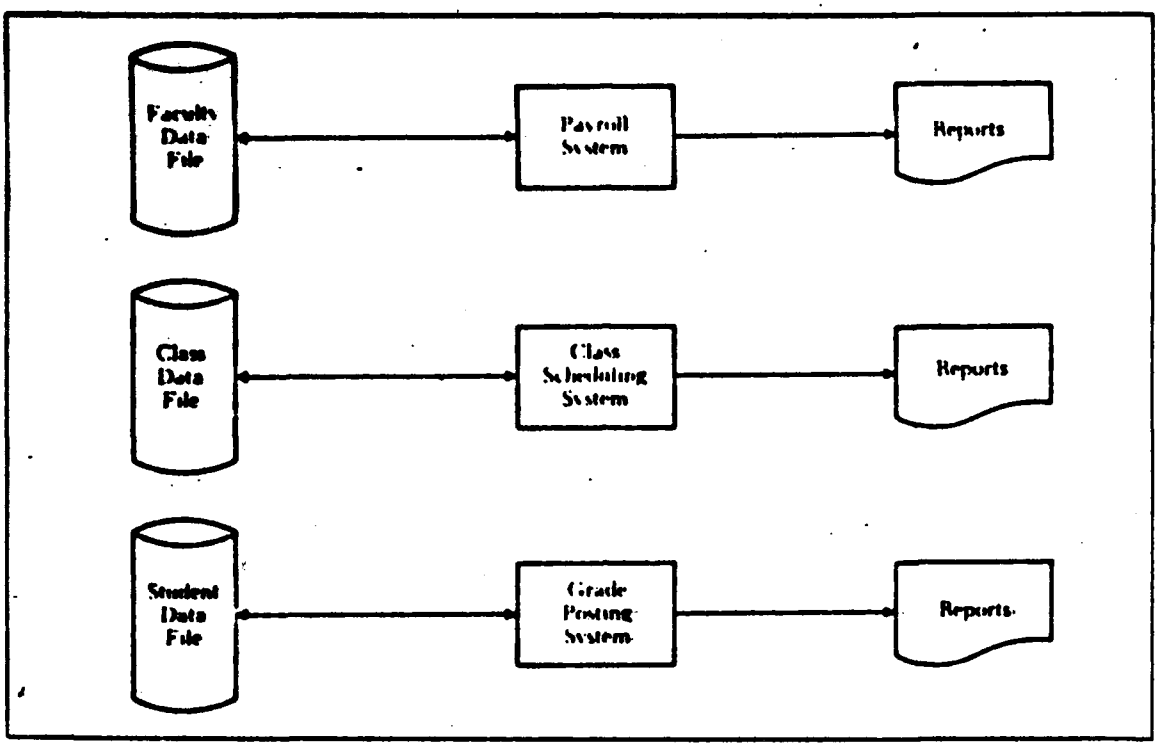


Figure 2.3 Example of File Processing Systems(pre-database)[2].

James Martin, a well known author in the data base field, defined a data base as

> a collection of interrelated data stored together without unnecessary redundancy to serve multiple applications. The data are stored so that they are independent of the programs which use them. A common and controlled approach is used in adding new data and modifying and retrieving existing data. The data are structured so as to provide a foundation for future application development. [Ref. 5: p.22]

Figure 2.3 and Figure 2.4 show the characteristics of database processing system.

---

[2]Kroenke, David M., *Database Processing: Fundamentals, Design, Implementation*, Science Research Associates INC., 1983, p.2

4. DBMS characteristics

The current science clearly demands an environment that is quite different and features flexibility and expandability - inherent features in a data base management system. Specifically, DBMS possesses the following characteristics:

(1) Data shareability. Through the structuring of data bases and control of the DBMS, data may be shared among many independent computer applications. Since some applications may not be planned until well after the completion and installation of other applications that use the same data bases, computer programs must have the ability to use commonly available data. Provision is therefore made for multiple uses of the same data, but control is exercised over access and over interfaces between independent programs.

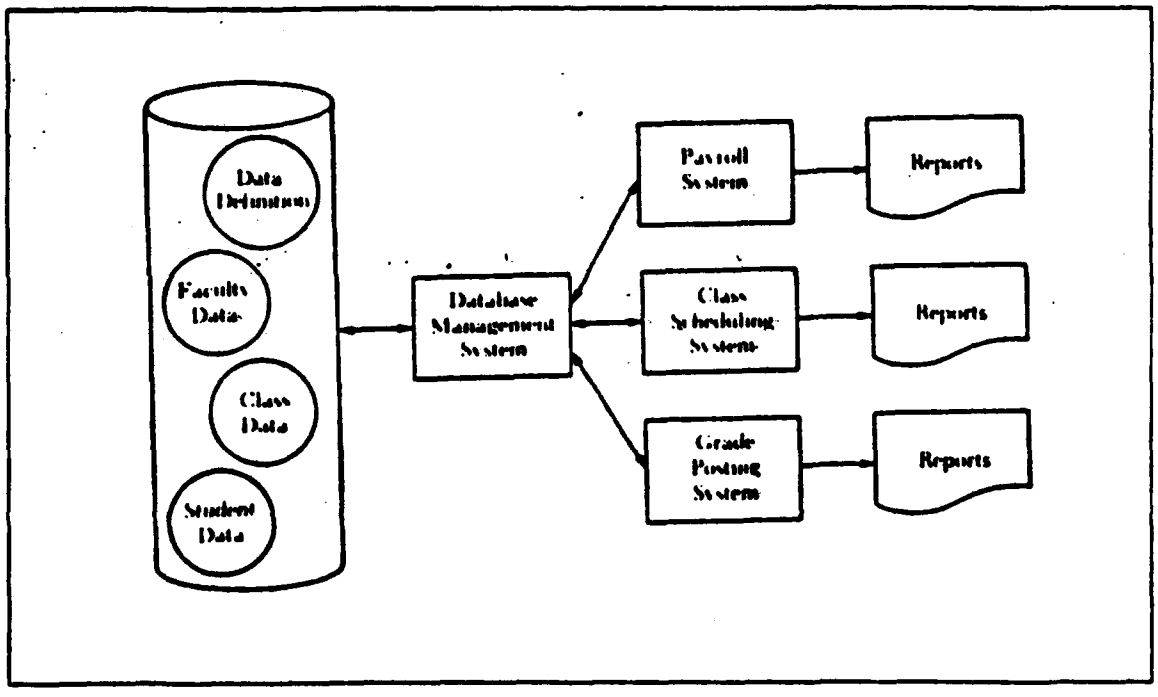


Figure 2.4 Example of Data Base Processing System[3].

(2) Data independence. A seemingly trivial application design change or database expansion can have an undesirable cost impact. Hence, it is important to avoid dependence of computer programs on a fixed physical data base. DBMS permits the addition and deletion of the data base, data fields or data records without modifying existing computer programs.

(3) Control of data redundancy. proliferation of similar data as more and more data bases are integrated. While not eliminating all redundancy, DBMS avoids much of this by structuring common data needs is terms of so-called logical data relationships that avoid duplicate data values.

---

[3]Kroenke, David M., *Database Processing: Fundamentals, Design, Implementation*, Science Research Associates INC., 1983, p.4

(4) Data integrity and security. In as many users share the same data files, DBMS maintains control over the integrity of the data base through synchronization of updates, insuring validity of data, examining the propagation of changes to data item values or to dependent data items, and maintaining an audit trial of interfaces between programs and data. Data security measures must also be applied to all DBMS structures, recognizing that accessibility to data should be a function of the sensitivity of the data, the processing procedures, and the authority of the user. Timing intervals for duplicating all data bases must be specified and provision made for on-promise as well as off-promise storage for back up purposes.

Database technology allows an organization's data to be processed as an integrated whole. It reduces artificiality imposed by separate files for separate applications and permits users to access data more naturally. Data integration offers several important advantages:

- More information from the same amount of data
- New reports and one-a-kind requests more easily implemented
- Elimination of data duplication
- Program/Data independence
- Better data management
- Affordable,sophisticated programming
- Representation of record relationships

On the other hand, database processing has a few disadvantages:

- Expensive database management system
- Higher operating cost
- Complexity
- Recovery is more difficult
- Increased vulnerability to failure

5. Levels of abstraction in a DBMS

A fairly standard viewpoint regarding levels of abstraction is shown in Figure 2.5. [Ref. 6: p.3] There we see a single database, which may be one of many databases using the same DBMS software, at three different levels of abstraction. The conceptual database is an abstract representation of the physical database (or, equivalently, we may say the physical database is an implementation of the conceptual database), and the views are each abstraction of portions of the conceptual database. The difference in the level of abstraction between views and the conceptual database is generally not great.

The term scheme is used to refer to plans, so we talk of a conceptual scheme as the plan for the conceptual database, and we call the physical database plan a physical scheme. The plan for a view is often referred to simply as a subscheme.

a. The conceptual scheme and its model

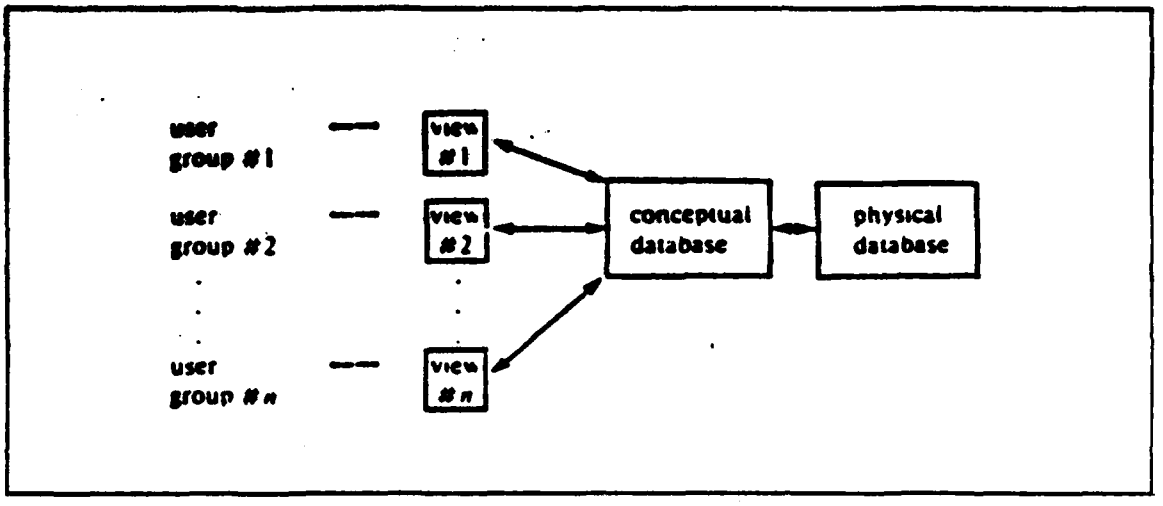


Figure 2.5 Levels of Abstraction in A Database System[4].

As we have said, the conceptual scheme is an abstraction of the real world pertinent to an organization like enterprise. A DBMS provides a data definition language to specify the conceptual scheme and, most likely, some of the details regarding the implementation of the conceptual scheme by the physical scheme. The data definition language is a high-level language, enabling one to describe the conceptual scheme in terms of a "data model".

b. Views

A view or subscheme is an abstract model of a portion of the conceptual database or conceptual scheme. For example, an airline may provide a computerized reservation service, consisting of data and a collection of programs that deal with flights and passengers. These programs, and the people who use them, do not need to know about personnel files or the assignment of pilots to flights. The dispatcher may need to know about flights, aircraft, and aspects of the personnel files (e.g., which pilots are qualified to fly a 747), but does not need to know about personnel salaries or the passengers booked on a flight.

[4]Ullman, Jeffery D., *Principles of database system*, Computer Science Press INC., 1980, p.3

c. The physical database

At the lowest level of abstraction with which we deal, there is a physical database. The physical database resides permanently on secondary storage devices, such as disks and tapes. We may view the physical database itself at several levels of abstraction, ranging from that of records and files in a programming language such as PL/I, perhaps through the level of logical records, as supported by the operating system underlying the DBMS, down to the level of bits and physical addresses on storage devices.

6. The objectives of database system organizations

A database system should be the repository of the data needed for an organization's data processing. That data should be accurate, private, and protected from damage. The system should be designed so that diverse applications with different data/information requirements can employ the data. Different end-users have different views of data which should be derived from a common overall data structure. In order to achieve these user requirements and others, the following objectives are considered by database system designers. [Ref. 7: p.34]

(1) The database is THE FOUNDATION OF FUTURE APPLICATION DEVELOPMENT. It should make application development easier, cheaper, faster, and more flexible.

(2) THE DATA CAN HAVE MULTIPLE USES. Different users who perceive the same data differently can employ them in different ways.

(3) CLARITY. Users can easily determine and understand what data are available to them.

(4) EASE OF USE. Users can gain access to data in a simple fashion. Complexity is hidden from the users by the DBMS.

(5) FLEXIBLE USAGE. The data can be used or searched in several ways with different access paths.

(6) CHANGE IS EASY. The database can grow and change without interfering with established procedures for using the data.

(7) LOW COST. The cost of storing and using data, and the cost of making changes, must be as small as possible.

(8) LESS DATA PROLIFERATION. New application needs may be met with existing data rather than creating new files, thus avoiding the excessive proliferation in today's tape libraries.

(9) PERFORMANCE. Data requests can be satisfied with speed suitable to the usage of the data.

(10) PRIVACY. Unauthorized access to the data will be prevented. The same data should be restricted in different ways from different uses.

(11) AVAILABILITY. Data should be available to users at the time when they need them.

(12) **RELIABILITY.** Almost all information/data for personnel management is very important to both individual personnel (e.g. for promotion, for new assignment, etc.) and group personnel. The information which is derived from database processing must be very reliable.

D. DATABASE MODEL

A data model is a collection of data structures together with a collection of operations that manipulate the data structures for the purpose of storing, querying, or processing the structure contents. A data model may also include the integrity constraints defined over the data structures, or it may include access control facilities of mechanisms for defining various external user views of the database. Some data models provide physical storage structures and physical access methods as part of the data model, but usually a data model is limited to the data structures and operations that are available to an end user and may be accessed from an application program. There are two reasons for studying database models. First, they are an important database design tool. Database models can be used for both logical and physical database design - much as flow charts or pseudocode are used for program design. Second, database models are used to categorize DBMS products. In this section, we will discuss the components of a database model and survey six important models.

1. Component of database model

Database models have two major components. The Data Definition Language (DDL) is a vocabulary for defining the structure of the database. The DDL must include terms for defining records, fields, keys, and relationships. In addition the DDL should provide a facility for expressing database constraints. Data Manipulation Language (DML) is the second component of a database model. The DML is a vocabulary for describing the processing of the database. Facilities are needed to retrieve and change database data. There are two types of DML, procedural DML and nonprocedural DML. Procedural DML is a language for describing actions to be performed on the database. Procedural DML obtains a described result by specifying operations to be performed. For procedural DML, facilities are needed to define the data to be operated on and to express the actions to be taken. Both data items and relationships can be accessed or modified. Nonprocedural DML is a language for describing the data that is wanted without describing how to obtain it.

2. Prominent database models.

Figure 2.6 portrays six common and useful database models. Models on the left-hand side of this figure tend to be oriented to humans and human meaning,

whereas those on the right-hand side are more oriented toward machines and machine specifications.

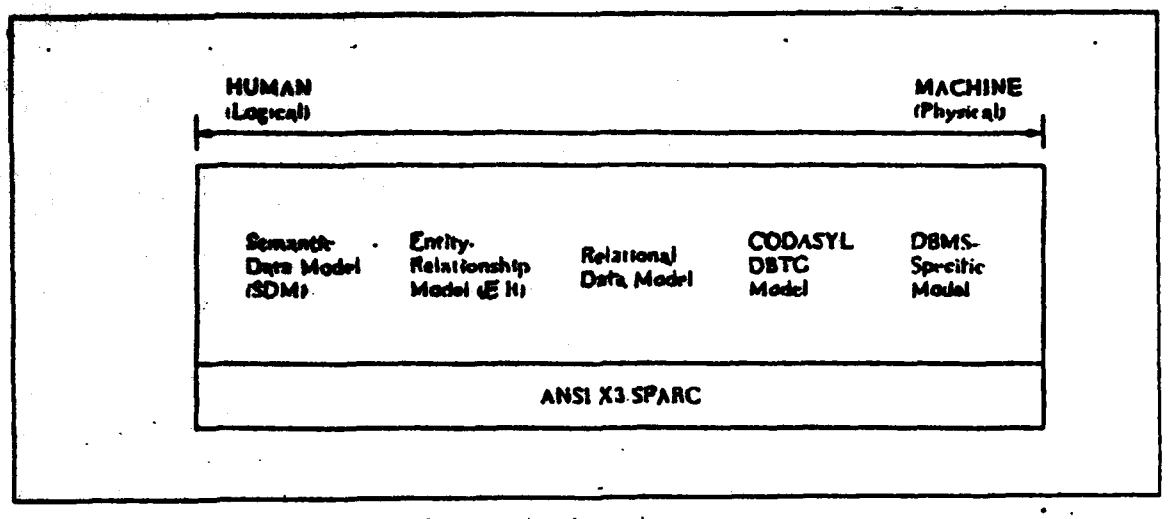


Figure 2.6 Relationship of Six Important Data Models.

a. Relational data model

The relational database model is near the midpoint of the human/machine continuum in Figure 2.9, because it has both logical and physical characteristics. The relational model is logical in that data are represented in a format familiar to humans; the relational model is unconcerned with how the data are represented in computer files. On the other hand, this is more physical than SDM(semantic data model) or the E_R(entity relationship) model. Database that have been designed according to the relational need not be transformed into some other format before implementation. Thus the relational model can be used for both logical and physical database design. A relation is simply a flat file. The rows of the relation are the file records. Rows are sometimes called tuples of the relation. The field of the relation (in the columns) are sometimes called the attributes of the relation. The significance of the relational model is not that data are arranged in relations but that relationships are concerned to be implied by data values. The principle advantage of carrying relationships in data is flexibility. Relationships need not be predefined. [Ref. 7: p.196]

b. Semantic data model

The word semantic means meaning. The semantic data model provides a vocabulary for expressing the meaning as well as the structure of database data. As

such, SDM is useful for logical database design and documentation. SDM provides a precise documentation and communication medium for database users. In particular, a new user of a large and complex database should find its SDM schema of use in determining what information is contained in the database. Also, SDM provides the basis for a variety of high level semantics-based user interfaces to a database. SDM has been designed to satisfy a number of criteria that are not met by contemporary database models. The chief advantage of SDM is that it provides a facility for expressing meaning about the data in the database. Another advantage of SDM is that it allows data to be described in context. Users see data from different perspectives. They see it relative to their field of operation. SDM allows relative data definition. A third advantage of SDM is that constraints on database data can be defined. For example, if a given item is not changeable, SDM allows this fact to be stated. With other data models, such constraints are not part of the schema description and are documented separately. SDM is like a pseudocode. But,instead of describing the structure of programs as pseudocode does,SDM describes the structure of data. Like pseudocode, SDM has certain structures and rules, and within those structures and rules, the designer has a good deal of latitude and flexibility. [Ref. 7: p.193]

c. Entity - Relationship model

The entity- relationship model(E-R model) is primarily a logical database model, although it has some aspects of a physical model as well. As its name implies, the E-R model is explicit about relationship. Unlike SDM, in the E-R model both entities and relationships are considered to be different constructs. Entities are grouped into entity sets, and relationships are grouped into relationship sets. An entity-relationship diagram is a graphical portrayal of entities and their relationships. It is useful to summarize the information in a design. It supports the representation of more general relationships. [Ref. 7: p.194]

d. CODASYL DBTG model

The CODASYL DBTG(conference on Data System Languages, Database Task Group) data model was developed by the same group that formulated COBOL during the late 1960s and is the oldest of the data models. The DBTG model is a physical database model. There are constructs for defining physical characteristics of data, for describing where data should be located, for instructing the DBMS regarding what data structures to use for implementing record relationships, and other similar

physical characteristics. A DBTG schema is the collection of all records and relationships. A subschema is a subset and reordering of records and relationships in the schema. Unlike the relational model, relationships become fixed when they are defined in the schema. Several reasons account for the lukewarm response that the CODASYL model has received, including the fact that it has a decidedly COBOL flavor to it. Finally,although most of the core concepts of the model are defined and agreed upon, there are many not-agreed-on variants of the core concepts. These variants create confusion and lead to a dilemma. [Ref. 7: p.197]

e. DBMS-specific models

There are over one hundred different commercial DBMS products. The DBMS are sometimes categorized in terms of their underlying data model. A DBMS is considered a relational system if it conforms, in essence, to the relational data model. Alternately, a DBMS is considered to be a CODASYL system if it conforms, in essence, to the CODASYL DBTG data model. A third category of DBMS is other. If a DBMS does not conform to one of the above two data models, then it has its own, unique data model. There are many systems that fall into the other category. [Ref. 7: p.198]

f. ANSI/X3/SPARC data model

The ANSI/X3/SPARC(American National Standards Institute / Committee X3 / Standards Planning and Requirements (sub)-Committee) data model does support a variety of different data models in Figure 2.6. This model is a model for DBMS design rather than for database design. This have the external, conceptual, and internal schema. [Ref. 7: p.198]

III. RELATIONAL MODEL

A. INTRODUCTION

A relation is a mathematical term for a two-dimensional table. It is characterised by rows and columns, each entry there being a data item value. The reason for calling this a relation rather than a matrix lies in the lack of homogeneity in its entries - the entries are homogeneous in the columns but not in the rows. A relational data base consists of such relations, which can be stored on a physical device in a variety of ways.

From the late 1960s a number of people toyed with the idea of constructing data base with relations as the basic building blocks. Most of these early systems were restricted to relations with only two columns, and all of them were special-purpose models incapable of meeting general data-processing requirements. In 1970 E.F. Codd of IBM proposed a model for a generalised relational Data Base System chiefly to provide data independence and data consistency, which are difficult to achieve in the formatted Data Base Systems. The model was subsequently improved and expanded by Codd and is now regarded by many as the future of all Data Base Systems. Needless to say, the term relational data base or relational model is nowadays generally indentified with Codd's model alone.

A basic feature of the relational model is its simplicity. A relation is a table of data and it may consist of only one row and one column, thus providing the simplest possible data structure which can be used as the common denominator of all data structures. It simplifies the design of the schema since there is only one logical data structure-the relation-to consider, without having to worry about the construction of the right data structures to represent complex data relationships. Furthermore the relational model provides an unparalleled freedom to the application programer by enabling him to access any data item value in the data base directly, the access mechanism being associative or content addressable since a data item is accessed directly by its value rather than by its relative position or by a pointer.

The concepts of the relational model are founded on mathematics, and all the terms used are mathematical. This has the effect of scaring off most people who would normally be interested in a data base.

In this chapter we shall keep the involvement with mathematics to a minimum. All concepts will be defined in non-mathematical terms in a simplified manner, sacrificing in the process some of the mathematical rigour which is really unnecessary for the understanding of the model. we shall also give the data-processing equivalent of the commonly used relational concepts.

B. BASIC CONCEPT

1. Terminology

A relation is simply a two-dimensional table that has several properties. First, the entries in the table are single-valued; neither repeating groups nor arrays are allowed. Second, the entries in any column are all of the same kind. For example, one column may contain person numbers, and another ages. Further, each column has a unique name and the other of the columns is immaterial. Columns of a relation are referred to as attributes. Finally, no two rows in the table are identical and the order of the rows is insignificant. Figure 3.1 portrays a relation.

Name	MSN	Rank	Age
Jae B. Park	667423	Capt	30
Sam N. Kim	651242	Maj	33
Young S. Jung	652310	Maj	34
Tae H. Choi	632258	Col	44

Figure 3.1 Korean Military Person Relation.

Each row of the relation is known as a tuple. If the relation has n columns or n attributes is said to be of degree n. The relation in Figure 3.1 is of degree 4, and each row is a 4-tuple.

Each attribute has a domain, which is the set of values that the attribute can have. For example, the domain of the Rank attribute in Figure 3.1 is the three values, Capt, Maj and Col. The domain of the Age attribute is all positive integers less then,say,100.

A relation of degree n has n domains, not all of which need be unique. For example, the relation in Figure 3.2 has age and age of spouse attributes. The domains of the two attributes are the same, integers from 1 to 100. To differentiate between attributes that have the same domain, each is given a unique attribute name. The attribute names for the relation in Figure 3.2 are Name, Position, Spouse-name, Age, Spouse-age, and Unit.

Name	Position	S_name	Age	S_age	Unit
Jae B. Park	A	Eun K. LEE	30	25	212SQ
Sam N. Kim	B	ki O. Sin	33	28	117SQ
You S. Jung	C	Hye S. Lee	34	29	808SQ
Tae H. Choi	D	Myen J. Cho	44	42	512SQ

Figure 3.2 Korean Air Force Pilot Relation.

Figures 3.1 and 3.2 are examples, or occurences. The generalized format, KOREAN MILITARY PERSON (Name, MSN, Rank, Age), is called the relation structure, and is what most people mean when they use the term relation. If we add constraints on allowable data values to the relation structure, we then have a relational schema. [Ref. 7: pp243,245]

As mentioned earlier, a relation is a mathematical term used to define a special kind of table. Each column is called a domain containing all the values of an attribute, and each row a tuple. The word tuple is taken from the description of groups, such as quintuple and sextuple. Thus a group of n elements is an n-tuple. In a relation of n domains, each tuple, that is, each row, is an n-tuple. The number of rows or tuples in a relation is its cardinality, and the number of columns is its degree. The individual elements in a relation are attributes values. If we consider an m × n relation (m rows and n columns), we have.

> a relation of degree n and cardinality m, that is, a relation containing n domains and n tuples, each tuple being an n-tuple.. There are m × n attribute

values, each tuple having n columns or n attribute values.

A relation of degree 1 is called unary, degree 2 binary, degree 3 ternary and degree n n-ary relation.. The characteristics of a relation are as follows.

(1) All entries in a domain are of the same kind.

(2) Domains are assigned distinct names called attribute-names.

(3) The ordering of the domains is immaterial.

(4) Each tuple is distinct, that is, duplicate tuples are not allowed.

(5) The ordering of the tuples is immaterial. [Ref. 8: p.132]

2. Attribute and domain names

A domain ,unlike a tuple, can be duplicate. A domain name is the common name for identical domains whereas an attribute name is the unique name for an individual domain. Attribute names for identical domains are constructed from the common domain name by attaching suitable prefixes to it. Consider, for instance, a relation called HIERARCHY containing two identical domains - one for the superior unit number and the other for subordinate unit number - both holding the same type of information, that is,. unit number codes. If we assume a common domain name, UNIT, then we can construct two attribute names, SUP-UNIT for the superior unit numbers, and SUB-UNIT for the subordinate unit numbers. Unit code, for instance, is DIV for division, IRE for infantry regiment, ARE for artillery regiment, IBA for infantry battalion, ABA for artillery battalion etc. Using QUANTITY as the attribute name for the third domain which contains the numbers of a subordinate unit numbers present in its superior unit number, we can represent the triplet as shown in Figure 3.3.

From the mathematical point of view, a domain can be simple or non simple, a simple domain containing a single attribute and a nonsimple domain containing a repeating group or a multiple of attributes. Therefore the name of a simple domain can be identical with that of its attribute. A nonsimple domain can be broken down into simple domains, giving each a unique attribute name as we have done in the example above.

3. Keys and attributes

A tuple is identified by its key, constructed from a combination of one or more attributes so that no attribute there is redundant. A tuple can have more than one possible key, each of which can uniquely identify the tuple. All these possible keys are known as the candidate keys. One of these keys is arbitrarily selected to identify

HIERARCHY (	SUP_UNIT	SUB_UNIT	QUANTITY)
	DIV 102	IRE 572	6
	DIV 104	ARE 337	3
	ARE 337	ABA 325	5
	IRE 572	IBA 153	5

Figure 3.3 A Relation of Degree 3.

the tuple and this key is known as the primary key. For example, consider a tuple with the following attributes

Division Code, Regiment Code, Regiment Commander No., No. of people.

If we assume that every regiment has its own separate commander, then this tuple can be uniquely identified either by Division Code + Regiment Code , or Division Code + Regiment Commander No. These then are two candidate keys, one of which can be selected as the primary key. Since a key must not contain redundant attributes, the Regiment Code and Regiment Commander No. can not appear in the same key, because the Regiment Commander No. implicitly defines Regiment Code.

If a tuple has attributes whose combination is the primary key in another relation, then this combination is called a foreign key. For instance Division Code can be a foreign key. An attribute that forms a part of a candidate key is a prime attribute of the tuple. The other attributes are nonprime. In the example given above, the Division Code, Regiment Code and Regiment Commander No. are prime attributes, and the No. of people is a nonprime attribute.

4. **Comparison with standard data-processing concepts**

In data-processing terms we may approximate a relation to the occurrences of a record type, a tuple to a record occurrence, and an attribute to a data item, a domain being the collection of all values for a single data item. Degree is the number of data items in the record and cardinality is the total number of records in the record type. A unary record relation is a record type consisting of a single data item; a binary relation is a record type of two data items; and so on.

However, there are some differences between record types and relations in third normal form, a record type being equivalent to an unnormalised relation where repeating groups are permitted. Normalised form will be discussed in Chapter IV. The ordering of the data items-that is, their relative positions-is fixed in a record type and cannot be altered, but the domains of a relations are independent of their relative positions since they are addressed individually by their attribute names. In a relation, the ordering of tuples is also unimportant because each of them is accessed directly, but this is not generally true for the records of a record type, unless they are specifically stored for direct retrieval. These access advantages follow directly from the content-addressable accessing concept used in relations as mentioned earlier. Finally by definition a relation can have a duplicate tuple, but there is no such conceptual restriction on the existence of a duplicate record in a record type. These discussions are summerized in Figures 3.4 and 3.5. [Ref. 8: p.134]

Relational Terms	Data-processing Terms
Relation	All the occurence of a recorde type
Tuple	Record
Attribute	Data item
Domain	All the values of a data item
Degree	Number of data items in the record type
Cardinality	Total number of records in the record type

Figure 3.4 Equivalence of relational terms with data-processing concepts[5].

[5]Deen S. M., *Fundamentals of Database Systems*, p.134,Hayden book company INC.,Rochelle park, New Jersy 1973

Item	Relation	Record type
Repeating group	Not allowed in normalised relations	Allowed
Odering of domains or data items	Immaterial	Important
Ordering of tuples or records	Immaterial	Important
Duplicate tuple or record	Not Allowed	Immaterial

Figure 3.5 Difference between relation and data-processing concepts[6].

C. EXPRESSING RELATIONSHIPS WITH THE RELATIONAL MODEL

When we design a database, we may need to specify the logical relationships that will exist among data base records. When we consider the user's requirement, we should realize that there are three fundamental types of record relationships. These types are: tree(or hierarchy), simple network, and complex network.

Many relations which are based on those three types of record relationships will be discussed in this section. First of all, each type of relationship is:

(1) Trees or hierarchies. A tree is a collection of records and one-to-many relationships among records. According to standard terminology, the records are called nodes, and the relationship between the records are called branches. The node at the top of the tree is called the root. Every node of a tree has a parent-the node immediately above it. Figure 3.6 shows a tree relationship.

(2) Simple Networks. A simple network is also a collection of records and one-to-many relationships among records. In a simple network, a record may have more than one parent, as long as the parents are different types of

[6]Deen S. M., *Fundamentals of Database Systems*, p.134,Hayden book company INC.,Rochelle park, New Jersey 1973

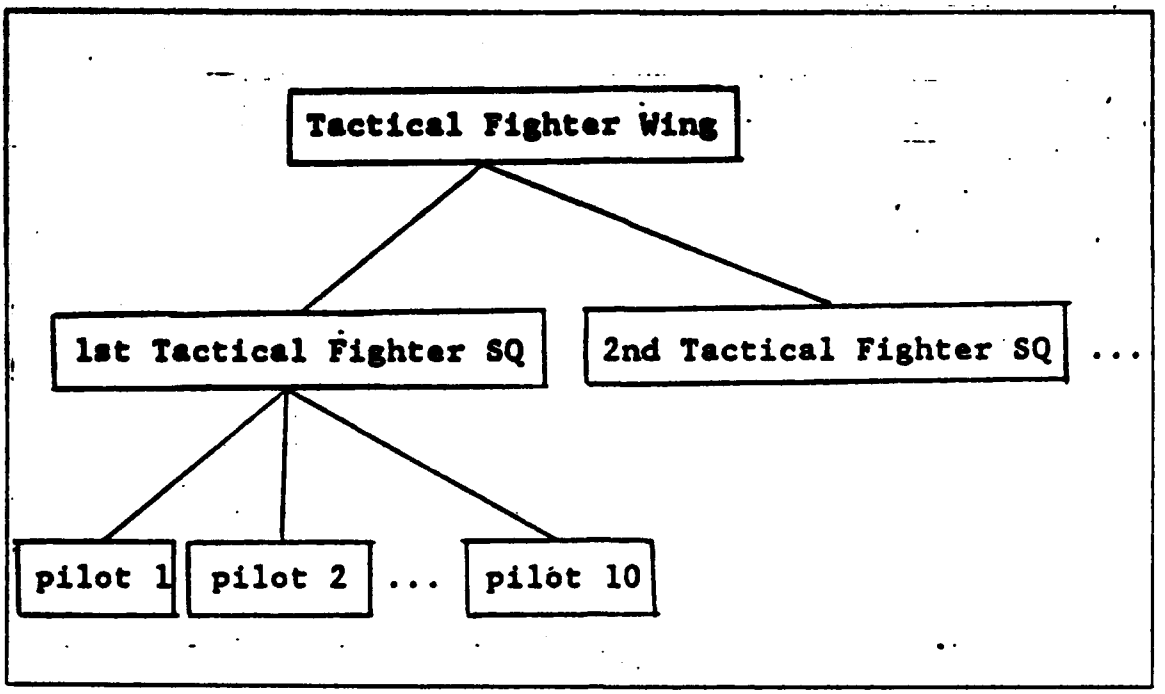


Figure 3.6 Example of Tree Relationship.

records. Figure 3.7 shows a simple network relationship.

(3) Complex Networks. A complex network is also a collection of records and relationships. The relationships are many-to-many instead of one-to-many. Figure 3.8 shows Complex Network Relationship.

1. Tree or hierarchical relationships

The tree which is illustrated in Figure 3.6 can be modeled by constructing two relations as in Figure 3.9.

Relation ML contains the Name, Mission, Location attributes, and relation PILOT contains the Name and Pilot attributes. Name is primary key of the ML relation, and Name and Pilot together are the primary key of the PILOT relation. This relational representation is useful when we need a information about pilots who work in specific wing or squadron.

2. Simple Network Relationships

Consider the undergraduate education/squadron/pilots relationship as it is shown in Figure 3.7. As we mentioned earlier, Figure 3.7 represents a simple network relationship. In Figure 3.10, the following relation structure will represent this network:

```
EDUCATION ( School, Major, Grad_year )
```

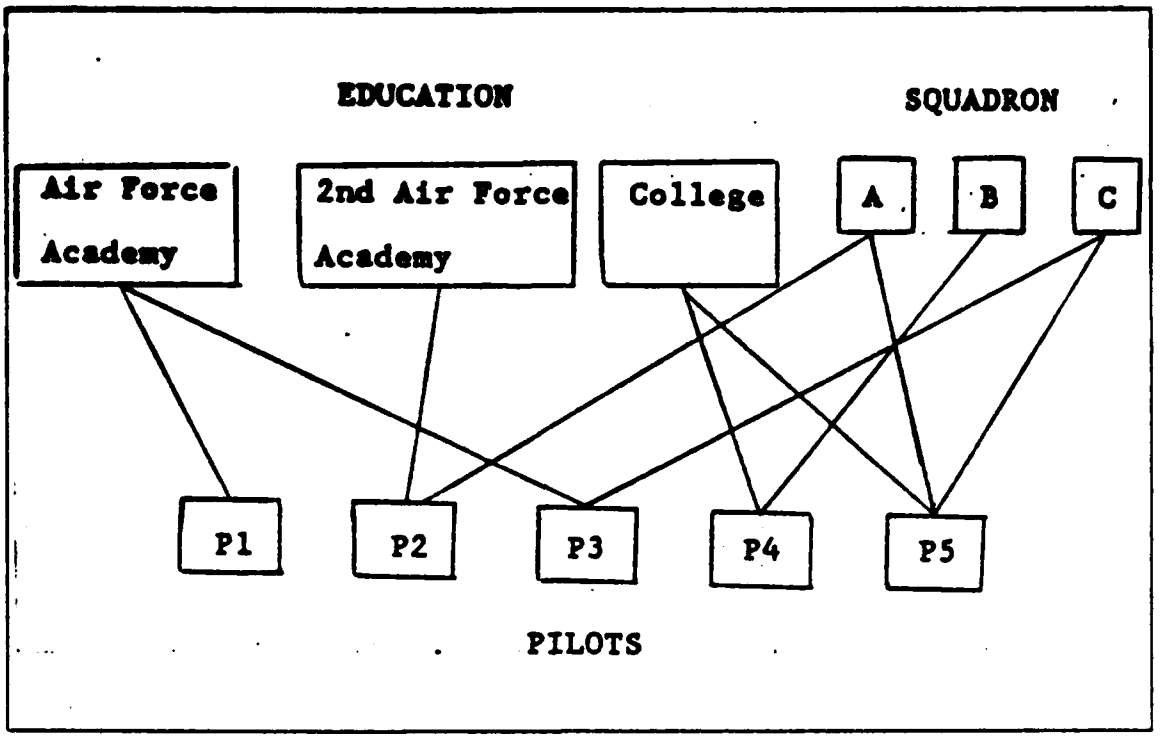


Figure 3.7 Example of Simple Network Relationship.

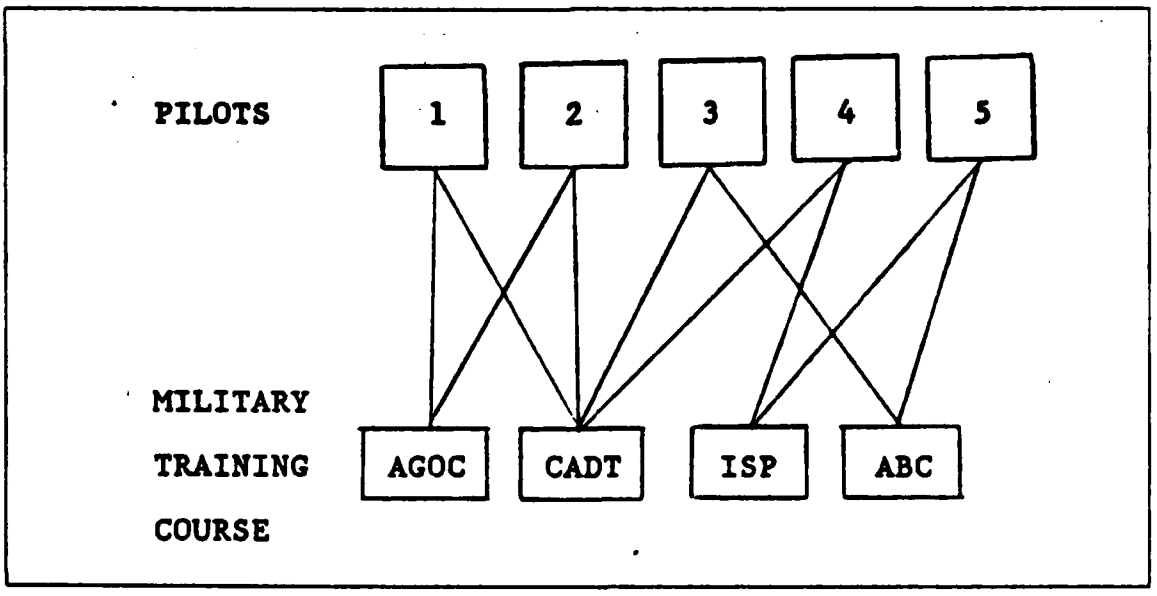
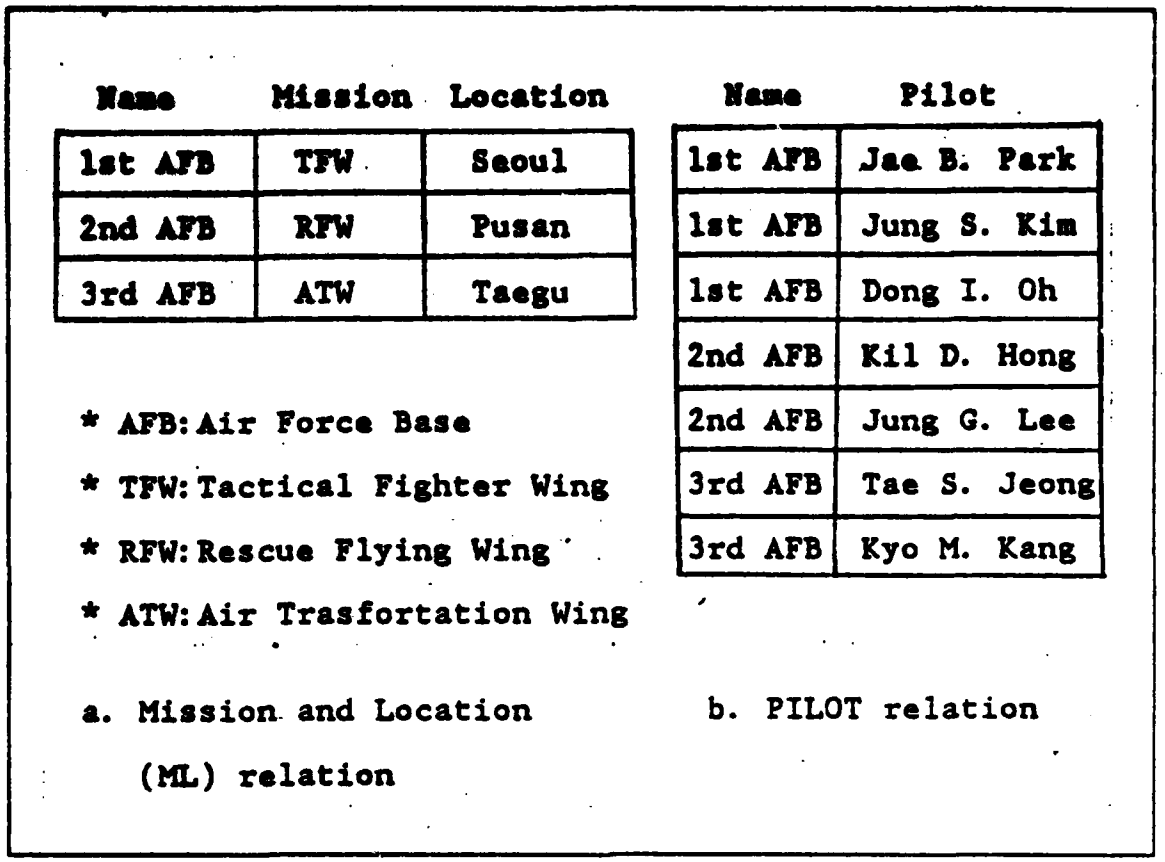


Figure 3.8 Example of Complex Network Relationship.

```
SQUADRON  (Name, Num_pilot )
```



Name	Mission	Location
1st AFB	TFW	Seoul
2nd AFB	RFW	Pusan
3rd AFB	ATW	Taegu

* AFB: Air Force Base
* TFW: Tactical Fighter Wing
* RFW: Rescue Flying Wing
* ATW: Air Trasfortation Wing

a. Mission and Location (ML) relation

Name	Pilot
1st AFB	Jae B. Park
1st AFB	Jung S. Kim
1st AFB	Dong I. Oh
2nd AFB	Kil D. Hong
2nd AFB	Jung G. Lee
3rd AFB	Tae S. Jeong
3rd AFB	Kyo M. Kang

b. PILOT relation

Figure 3.9 Relational Representation of Tree Relationship.

PILOT (Mil_serv_num, Name, School, Major, Squadron)

As it is shown in Figure 3.10, there are no special constructs to represent the relationship.

Instead, relationships can be determined by pairing equal values of attributes. These relations are very useful when we need some information about pilots who graduate from a specific school and study specific subjects. For example, let's assume that we want to know how many pilots work with Jae B. Park. To respond to this query, we can find PILOT tuples for Jae B. Park. Next, we can find the number of pilots from 333rd SQ tuple in the SQUADRON relation.

3. Complex Network Relationships

The relational model representation of a complex relationship is similar to that of simple network relationship. Figure 3.11 is based on Figure 3.8. A straightfoward way of representing this structure is to define three relations: one for

EDUCATION Relation

School	Major	G_year
AF Academy	Mechanical Eng.	1980
2nd AF Academy	O.R	1977
Air College	Management	1982

SQUADRON Relation

Name	N_pilot
333rd SQ	10
505th SQ	50

PILOT Relation

MSN	Name	School	Major	Squadron
61543	Jar Park	AF Academy	Mechanical Eng.	333rd SQ
65320	Gil Hong	2nd AF Academy	O.R	505th SQ
54252	Tae Lee	Air College	Management	270th SQ

Figure 3.10 Relational representation of Simple Network Relationships.

pilots, one for military training courses, and one for the relationship between pilots and military training courses. This last relation is an intersection record. The following relation structure will represent this network:

PILOTS (Mil_serv_num, Name, Rank)

MTC (Cour_name, Year, Num-student)

PILOT_MTC (Mil_serv_num, Cour_name).

It is very easy to find someone's career by using these relations. For example, let's consider the question "What kinds of military training courses has Capt. Jae Park taken so far?". According to Figure 3.11, he took AGOC and CADT course.

MSN	Name	Rank
61543	Jae Park	Capt
65320	Gil Hong	Capt
54252	Tae Lee	Col

PILOTS Relation

Cname	Year	No_Student
AGOC	83-7	50
ABC	82-5	60
CADT	83-8	70
ISP	84-3	80

MTC(military training - course) Relation

MSN	Cname
61543	AGOC
61543	CADT
65320	CADT
65320	ISP
54252	ABC
54252	ISP

PILOTS_MTC Relation

Figure 3.11 Relational representation of Complex Network Relationship.

To manipulate data in the database by the application program we need a Data Manipulation Language or DML- one for each host language. The DML acts as an interface language with the database. Its major functions are

- to select a record from the database
- to represent it to the application program
- to add new records and relationships in the database
- to change existing records and relationships in the database
- to remove existing records and relationships from the database

[Ref. 8: p.52].

As it is discussed in Figure 3.11, understanding of representations of three relationships are very important to get flexible and useful information in a personnel management system. Each relationship is useful for individual required characteristic of query. An example is shown in Figure 3.11.

D. DATA MANIPULATION LANGUAGES

The notation for expressing queries is usually the most significant part of a data manipulation language. The nonquery aspects of a relational data manipulation language, or "query language," are often straightfoward, being concerned with the insertion, deletion and modification of the tuples. On the other hand, queries, which in the most general case are arbitrary functions applied to relations, often use a rich, high level language for their expression.

Query languages for the relational model break down into two broad classes:

(1) Algebraic languages, where queries are expressed by applying specialized operators to relations, and

(2) Predicate calculus languages, where queries describe a desired set of tuples by specifying a predicate the tuples must satisfy. [Ref. 6: p.104]

1. Relational algebra

Relational algebra operates one or more relations and produces a new relation as the result. The operations are expressed in a system of notation and they can be used to retrieve information from one or more relations or to update a tuple of a relation. We shall describe here six operations of which the first three- union, intersection and difference are traditional set operations; the other three - projection, join and division - are less common. Relations can be manipulated using the opertators +, -, etc., in high school algebra to achieve a desired result. Relation algebra is hard to use, partly because it is procedural. That is, when using relational

algebra we must know not only what we want, but also how to get it. In high school algebra, variables represented numbers, and operations like +, -, ×, and/ operated on numeric quantities. For relational algebra, however, the variables are relations, and the operations manipulate relations to form new relations. For example, the operation + (or union) combines the tuples of one relation with the tuples of another relation. [Ref. 7: p.255]

a. Union

The union of set A with set B, denoted as A ∪ B, is the set of all objects without repetition. We only apply the union operator to relations of the same number of attribute, so all tuples in the result have the same number of attribute. Each attribute should have same domain. This can be used to insert a new tuple to a relation.

b. Intersection

The intersection of set a with set B, denoted as A ∩ B, is the set of all objects belonging to both A and B. So the intersection of two relations is a third relation containing common tuples. Again, the relations must be union compatible. This can be used to find a duplicate tuple between two relations.

c. Difference

The difference of set b from set a, denoted as A - B, is the set of all objects belonging to A but not to B. That is, the difference of two relations is a third relation containing tuples which occur in the first relation but not in the second. The relations must be union compatible. This can be applied to delete a tuple. To amend a tuple, we must first delete it with a difference operation and then insert the amended tuple by a union operation.

d. Projection

Projection is the selection of one or more named domains from a relation in a specified order, followed by the elimination of duplicate tuples from the resulting relation. (In fact in all operations used in relational algebra, duplicate tuples are removed since they are not allowed in a relation.) That is, the result of the projection is a new relation having the selected attributes. In other words, projection picks columns out of a relation. Since the result of projection is a relation, and since relations can not contain duplicate tuples, the redundant tuple is eliminated. Projection can also be used to change the order of attributes in a relation. We shall use the notation R [ABC] to denote the projection of domains A, B and C in that order from relation R.

e. Join

The join operation is a combination of the product, and projection operations. The join of relation A with relation B produces a relation R that consists of all the possible tuples obtained by concatenating each tuple of A with all tuples of B that have the same value under the common domain. A tuple of an original relation is excluded from the resultant relation if its value under the common domain is not shared by a tuple of the other relation. The resultant relation contains all the domains of both the original relations, the common domain appearing only once. Let's assume there are two relations A (SQUADRON PILOT) and B (PILOT MTC) as given below.

A(SQUADRON	PILOT)	B(PILOT	MTC)
255SQ	Jae B. Park	Jae B. Park	AGOC
303SQ	Dong I. Lee	Jae B. Park	CADT
506SQ	Jang K. Cho	Jang K. Cho	ISP
355SQ	Jae S. Jeong	Hwan S. Lee	ABC

Their join R is

R (SQUADRON	PILOT	MTC)
255SQ	Jae B. Park	AGOC
255SQ	Jae B. Park	CADT
506SQ	Jang K. Cho	ISP

For operational convenience, in all subsequent join operations, we shall assume the common domain to be the rightmost domain of the first relation and the leftmost domain of the second relation as it is shown above; this can be ensured if necessary by a suitable projection operation. Join in conjunction with the projection operation provides a very useful tool for the manipulation of relations.

f. Product

The product of two relations (cartesian product) is the concatenation of every tuple of one relation with every tuple of a second relation. The product of relation A (having m tuples) and relation B (having n tuples) has m times n tuples. The product is denoted A × B or A times B. [Ref. 7: p.257]

g. Division

We may divide a binary relation by a unary relation if the domain of the unary relation is also a domain of the binary relation. The result of such a division is a unary relation containing the uncommon domain of the binary relation is selected for

the resultant relation, if its associated entries in the common domain contain all the values of the divisor domain. Consider a binary relation DT and three unary relations DI, DJ and DK as given below.

DT (S#	P#)	DI (P#)	DJ (P#)	DK (P#)
S1	P1	P1	P1	P1
S1	P2		P2	P2
S1	P3			P3
S1	P4			
S2	P1			
S2	P3			
S2	P4			
S3	P1			
S3	P2			

Denoting division by/(slash) and the resultant relation by R, we have

DT/DI=R(S#)	DT/DJ=R(S#)	DT/DK=R(S#)
S1	S1	S1
S2	S3	
S3		

For operational convenience in division, as in join, we shall assume the rightmost (that is, the second) domain of the dividend as the common domain. All algebraic operations will be evaluated from right to left giving precedent to projection operation over join and division, the priority over projection being indicated by ordinary brackets( ).

Relational algebra can be used for a procedural language. It is extremely powerful and is device independent, since the queries are based on the values of the data items rather than their positions. However, the construction of an algebraic expression for a query is very tedious, even though the technique can quickly be learnt. In addition, the nature of a query is not obvious from the algebraic expression unless it is patiently worked out. These tend to increase the chances of errors in the queries. Relational calculus is designed to improve this situation. [Ref. 8: p.150]

2. Relational calculus

Relational calculus is one of the strategies for manipulating relations. It is a nonprocedural language for expressing what we want without expressing how to get it. In relational algebra the user specifies the detailed procedures for extracting

information, whereas in relational calculus the user defines what he wants, leaving the system to work out the procedure required. The expression of a relational calculus has two parts, a target list which consists of a list of the wanted elements separated by commas, and a logical expression, called a predicate or qualification, which defines the wanted elements in terms of the relations from which they are to be extracted. It is written in the form

```
Target list : predicate
```

to be interpreted as: extract the elements in the target list such that (or where) the predicate is true. [Ref. 8: p.152]

IV. RELATIONAL DATABASE DESIGN

A. INTRODUCTION

The combination of DBMS software, applications software, database implementation, and operating system/hardware environment brought together to provide information services for users is known as a database system. Although the technology for DBMS, operating system, and applications programming is well developed, little attention has been given to the effective use of these tolls with alternative database structures. Thus, the major problem facing the database administrator is not whether to use this technology, but how to use it effectively. This problem can be summarized by a number of issues that arise through the life cycle of an application:

(1) What are the user requirements, and how can they be expressed?

(2) How can these requirements be translated into an effective database structure?

(3) When should, and how can, the database structure be adapted to new and/or changing requirements?

The process of developing a database structure from user requirements is called database design. Many practitioners have argued that they are at least two separate steps in the database design process: the design of a logical database structure which is processible by the DBMS and describes the user's view of the data, and then selection of a physical structure that includes data representation or encoding, access methods, and physical clustering of data. Other than the logical/physical delineation, however, the overall structure of the design process has not been well defined, and even the logical/physical boundary has been open to considerable dispute. We wish to avoid this confusion by defining more concisely each step in the design process.

Database design is an intuitive and artistic process. There is no algorithm for it. Typically, database design is an iterative process; during each iteration, the goal is to get closer to an acceptable design. Thus a design will be developed and then reviewed. Defects in the design will be identified, and the design will be redone. This process is repeated until the development team and users can find no major defects. (Unfortunately, this does not mean the design will work; it simply means no one can think of any reason why it won't.)

The database is the bridge between people and hardware. As mentioned earlier, the characteristics of both people and hardware need to be considered. Consequently, database design is divided into two phases: logical design, where the needs of people are specified, and physical design, where the logical design is mapped into the constraints of particular program and hardware products. Figure 4.1 illustrates the flow of work in a typical database design project. User requirements are studied and a logical database design is developed. Concurrently, the preliminary design of database processing programs is produced. Next, the logical database and the preliminary program designs are used to develop the physical database design and the detailed program specifications. Finally, both of these are input to the implementation phase of the project.

This chapter will introduce theoretical logical, physical database. And then we will discuss about relation database design in detail and hypothetical Korean Army or Air Force's data will be used to show examples.

Practical sample application program is shown in Appendix A. [Ref. 7: pp.177,178]

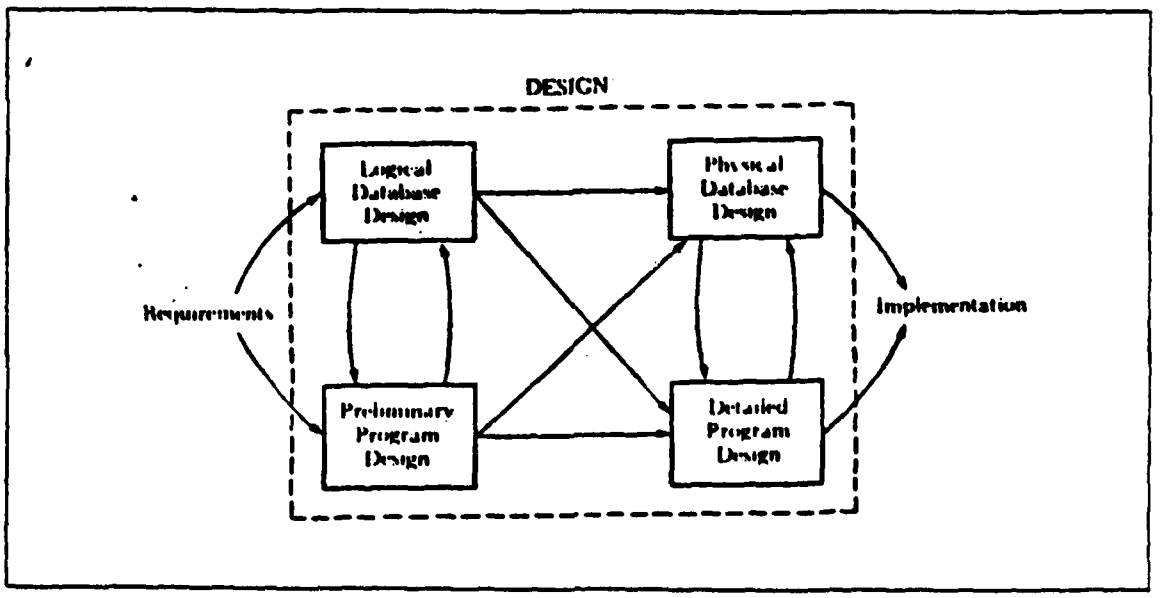


Figure 4.1 Database and Program Design Flow[7].

[7]Kroenke, David M., *Database processing: Fundamentals, Design, Implementation*, Science Research Associates INC.,1983

B. LOGICAL DATABASE DESIGN

Conceptual design deals with information independent of any actual implementation. It is the objective of conceptual design to represent information in a form that is comprehensible to the user independent of system specifies, but implementable on several systems.

1. Outputs of logical database design

A logical database design specifies the logical format of the database. The records to be maintained, their contents, and relationships among those records are specified. Industry uses various terms for this design. It is sometimes called the schema, the conceptual schema, or the logical schema. We will use the term logical schema because it is the schema developed during logical design.

a. Logical database records

To specify logical records, the designer must determine the level of detail of the database model. If the model is highly aggregated and generalized, there will be few records. If model is detailed, there will be many records. The database designer must examine the requirements to determine how coarse or how fine the database model should be. The contents of these records are specified during logical design. Figure 4.2 shows an example of field description.

b. Logical database record relationship

The essence of database is the representation of record relationships. These relationships are specified during logical design. The designer studies the application environment, examines the requirements, and identifies necessary relationships.

Figure 4.3 shows possible relationships for several records in military personnel management system's database. The arrows represent many-to-many relationships between database records. Data structure diagrams are not the only tool for expressing relationships. To summarize, their content, constraints, and relationships.

2. Inputs to logical database design

The inputs to logical database design are the system requirements and the project plan. Requirements are determined by interviews with users, and that they are approved by both users and management. The project plan describes the system environment, the development plan, and constraints and limitations on the system design.

Field	Description
PERSON Record	
Rank	Alphabetic, 25 character
Military_Serice_Number	Numeric, 15 decimal degit
Name	Alphabetic, 25 characters
Address	Alphabetic, 70 characters
MILITARY CAREER Record	
Military Service Number	Numeric, 15 decimal degit
Unit_name	Alphabetic, 20 characters
Duty _name	Alphabetic, 30 characters
Duty_rank	Alphabetic, 25 characters
Date	Format : YYMMDD

Figure 4.2 Sample field description for PERSON and MILITARY CAREER records.

The requirement will be expressed in the form of data flow diagrams, policy statements, and the data dictionary. Having the requirements in this form will greatly facilitate the logical design process. Contents of the data dictionary can be transformed into the logical and user's views. Policy statements can be used to develop the descriptions of logical database processing. The requirements can be used to verify the completeness of the logical design. If the requirements are defined in narrative style, they will need to be converted to a format that facilitates logical database design.

3. **Procedures for logical database design**

The major steps in the logical design process are described below.

(1) Identify data to be stored. The data dictionary is processed and data that is to be stored is identified and segregated.

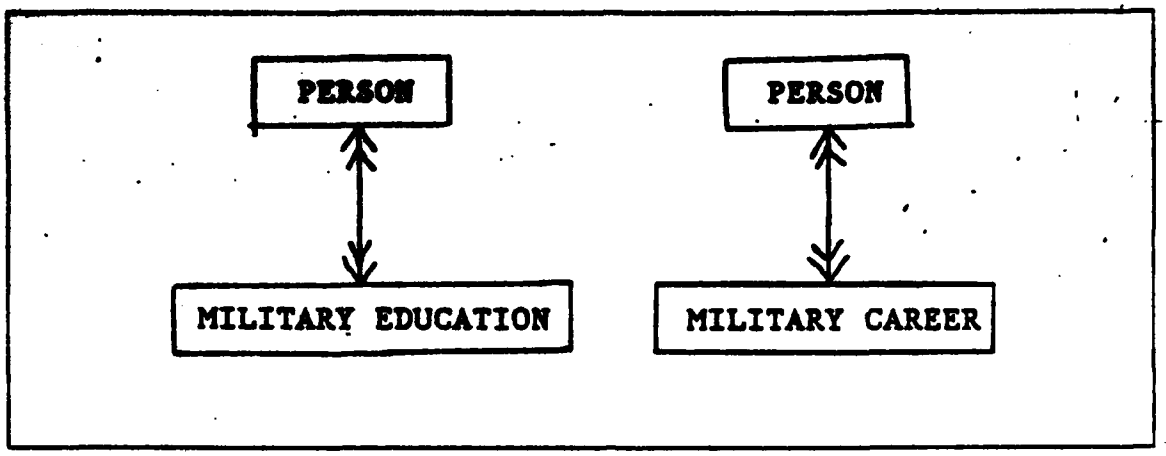


Figure 4.3 Data Structure Diagram.

(2) Consolidate and clarify data names. One task is to identify synonyms, to decide on standard names for synonyms, and to record aliases. Synonyms are two or more names for the same data item.

(3) Develop the logical schema. The third step in the design process is to develop the logical schema by defining records and relationships. Records are defined by determining the data items they will contain.

(4) Define processing. The requirements are examined to determine how the database should be manipulated to produce required results. The processing defines can be developed in several ways. One method is to describe transactions and data to be modified. Another method for describing database processing is to develop structure charts of the programs that will access the database. One method for developing such processing descriptions is called transform analysis.

(5) Design review. The logical schema and user views are examined in light of the requirements and program descriptions. Every attempt is made to identify omissions, unworkable aspects, or the flaws in the design. [Ref. 7: p.181]

C. PHYSICAL DATABASE DESIGN

The second stage of database design - physical design - is a stage of transformation. The logical schema is transformed into the particular data constructs that are available with the DBMS to be used. Whereas the logical design is DBMS - independent, the physical design is very much DBMS - dependent.

1. Outputs of physical database design

Specific constructs vary widely from one DBMS to another. At this point, we can not be very detailed. In general, two major specifications are produced. First, the physical specification of the logical schema is defined. We will call this specification the physical schema. This schema is transformation of the logical schema into the data modeling constructs available with the DBMS to be used. Second, user views are defined.

PHYSICAL SCHEMA. The content of each record must be defined, and the name and format of each field of each record specified. Constraints from the logical database design are transformed into criteria for field descriptions. Keys of database records need to be identified, and overhead structures for supporting the keys defined. For example, the designer may specify that a particular key is to be supported by an inverted list. Record relationships are also defined in the physical design. Limitations in the DBMS may necessitate that record relationships be changed from what the users wanted. A many-to-many relationship may need to be changed to a simple network, for example.

USER VIEWS. The second component of a physical database design is the user views. Since most users will need to view only a portion of the database, the logical design must specify which user groups will view which portions of the database.

User views are generally a subset of the schema. Records or relationships may be omitted from a view; fields may be omitted or rearranged. Also, the names of records, fields, or relationships may be changed. This flexibility allows uers to employ terminology that is familiar and useful to them. [Ref. 7: pp.188,189]

2. **Inputs to logical database**

The inputs to the physical database design are the outputs of the logical database design, the system requirements, and the preliminary design of programs.

3. **Physical design steps**

The physical design phase can also be categorized into distinct steps based on groups of related design decisions. However, once again, the proper ordering of these steps is open to conjecture, owing to the fairly strong dependencies between these groups of design decisions. Practical experience has shown that neither the starting point nor the order of steps can be definitely stated for a given design problem. On the other hand, the physical design phase can be regarded as an iterative process of initial design and refinement. Each of proposed steps needs to be performed several times, but each succeeding analysis should be done more quickly because the procedure is known and the number of unchanging performance variables should be increasing between iterations. Typically, two or three passes through the substeps will result in convergence to a solution. The relative importance of each step toward system performance becomes obvious through experience and careful documentation of the entire analysis.

The following five steps include three that represent the major categories of physical database structure design and two that involve constraints and program design.

STEP 1: STORED RECORD FORMAT DESIGN. Assuming that the logical record structure has been defined, this process addresses the problem of formatting stored data by analysis of the characteristics of data item types, distribution of their values, and their usage by various applications. Decisions on redundancy of data, derived versus explicitly stored values of data,and data compression are made here.

Certain data items are often accessed far more frequently than others, but each time a particular piece of data is needed, the entire stored record, and all stored records in physical block as well, must be accessed. Record partitioning defines an allocation of individual data items to separate physical devices of the same or different type, or separate extents on the same device, so that total cost of accessing data for a given set of user applications is minimized. Logically, data items related to a single entity are still considered to be connected, and physically they can still be retrieved together when necessary. An extent is a contiguous area of physical storage on a particular device.

STEP 2: STORED RECORD CLUSTERING. One of the most important physical design considerations is the physical allocation of stored records, as a whole, to physical extents. Record clustering refers to the allocation of records of different types into physical clusters to take advantage of physical sequentiality whenever possible. Analysis of record must take access path configuration into account to avoid access-time degradation due to new placement of records.

Associated with both record clustering and record partitioning is the selection of physical block size. Blocks in a given clustered extent are influenced somewhat by stored record size, but also by storage characteristics of the physical devices. Furthermore, larger blocks are typically associated with sequential processing and smaller blocks with random processing. Thus, we see that although block size is closely related to clustering, it is also dependent on access path, type of applications, and hardware characteristics. Consequently, choice of block size may be subject to considerable revision during an iterative design process.

STEP 3: ACCESS METHOD DESIGN. An access method provides storage and retrieval capabilities for data stored on physical devices, usually secondary storage.

The two critical components of an access method are storage structure and search mechanisms. Storage structure defines the limits of possible access paths through indexes and stored records, and the search mechanisms define which paths are to be taken for given applications. Intrarecord design and device allocation aspects of storage structure are not used here, whereas index design and interrecord connections are quite relevant.

An attribute is an item type it may be used as a primary key, secondary key, or nonkey. A primary key uniquely defines a record. A secondary key is an attribute used as an index to records, but it does not uniquely identify those records. A nonkey is an attribute that is not used as a primary or secondary key for indexing or other search mechanism for records.

Access method design is often defined in terms of primary and secondary access path structure. The primary access paths are associated with initial record loading, or placement, and usually involve retrieval via the primary key. Individual files are first designed in this manner to process the dominant application most efficiently. For the same reason, physical databases may require several primary access paths. Secondary access paths include interfile linkages and alternate entry-point access to stored records via indexes on secondary keys. Access time can be greatly reduced through secondary indexes, but at the expense of increased storage space overhead and index maintenance. Step 1 - 3 are controlled by our DBMS.

STEP 4: INTEGRITY AND SECURITY CONSIDERATIONS. As in implementation design, trade-offs among integrity, security, and efficiency requirements must be analyzed.

STEP 5: PROGRAM DESIGN. The goal of physical data independence, if met, precludes application program modifications due to physical structure design decisions. Standard DBMS routines should be used for all accessing, and query or update transaction optimization should be performed at the systems software level. Consequently, application program design should be completed when the logical database structure is known. When physical data independence is not guaranteed, program modification is likely. For example, a program based on a navigational access method would have to be radically changed if entry-point access methods were introduced for the first time during the physical database design phase.

Design decisions are also required in other areas, many of which are quite system dependent. Some examples are selection of buffer pool size, redundancy of

stored records, and differential files. These issues appear to be equally important and difficult to analyze for both physical database structure and file design. [Ref. 9: pp.169,170]

D. NORMALISATION

1. Introduction

By now we have examined several aspects of database systems in general and relational systems in particular. But we have not yet considered a very fundamental question, namely: Given a body of data to be represented in a database, how do we decide on a suitable logical structure for that data? In other words, how do we decide what relations are needed and what their attributes should be? This is the database design problem. The topic of this section, normalisation theory, is basically a formalisation of simple ideas such as this one--a formalisation that has practical application in the area of database design.

Before going any further, we should stress the fact that designing a database can be an extremely complex task. Normalisation theory is a useful aid in design process, but it is not a panacea. Anyone designing a relational database is advised to be familiar with the basic techniques of normalisation as described in this section, but we certainly do not suggest that the design should be based on normalisation principles alone.

2. Functional dependence

A functional dependency is a relationship between attributes. Attribute Y is said to be functionally dependent on attribute X if the value of X determines the value of y. Another way of saying this is that if we know the value of X, we can determine the value of Y.

For example, as it is shown in Figure 4.4, attributes NAME and BIRTH of relation BASIC are each functionally dependent on attribute MSN because, given a particular value for MSN, there exists precisely one corresponding value for each of NAME, BIRTH and RANK. In symbols, we have

```
PERSON.MSN → PERSON.NAME
PERSON.MSN → PERSON.BIRTH
PERSON.MSN → PERSON.RANK
PERSON.MSN → PERSON.SPECIALTY
```

or, more succinctly

```
PERSON.MSN → PERSON.(NAME,BIRTH,RANK,SPECIALTY).
```

PERSON(MSN,Name,Birth)

key : MSN

MSN	NAME	BIRTH	RANK	SPECIALTY
166753	Jae Park	85.3.23	Capt.	pilot
166720	Sin Yang	85.6.12	Capt.	supply
166542	Jung Kim	85.2.20	Maj.	security

ASSIGNMENT(Name,Unit,Position,S_date)

key : UNIT, POSITION

NAME	UNIT	POSITION	START_DATE
Jae Park	555SQ	SQ commander	86.10.1
Sam Kim	554SQ	Intelligence officer	86.5.4
Gun Hong	553SQ	Operation officer	85.12.23

FAMILY(MSN,S_Name,S_SSN,NOD)

key : MSN

MSN	SPOUSE NAME	SPOUSE SSN	NO OF DEPENDENT
166753	Kyung Bang	1111-222	3
166720	Eun Park	2222-333	4
166542	Mi Chun	3333-444	2

Figure 4.4 Relational view.

The statement "PERSON.MSN → PERSON.NAME" is read as "attribute PERSON.NAME is functionally dependent on attribute BASIC.MSN", or, equivalently, "attribute PERSON.MSN functionally determines attribute PERSON.NAME". We also introduce the concept of full functional dependence. Attribute Y is fully functionally dependent on attribute X if it is functionally dependent on X and not functionally dependent on any proper subset of X. For example, in the relation FAMILY, the attribute NOD is functionally dependent on the composite attribute (MSN, SPOUSE NAME); however, it is not fully functionally dependent on this composite attribute because, of course, it is also functionally dependent on MSN alone.

On the other hand, attribute NOD is fully functionally dependent on the composite attribute (MSN, S_SSN).

The objectives of normalisation are:

- to make it feasible to represent any relation in the database
- to obtain powerful retrieval algorithms based on a simpler collection of relational operations than would otherwise be necessary
- to free relations from undesirable insertion, update, and deletion dependencies
- to reduce the need for restructuring the relations as new types of data are introduced
- to make the collection of relations neutral to the query statistics, where these statistics are liable to change as time goes by.

[Ref. 10: p.47]

3. **Normal form**

Normalisation theory is built around the concept of normal forms. A relation is said to be in a particular normal form if it is satisfies a certain specified set of constraints. The real world with entities and their properties displays a multitude of entity relationships which can be expressed in the form of two-dimensional tables or relations. These relations will in general be unnormalised, that is, they may contain repeating groups whose presence creates serious access problems leading to reduction in data independence. A relation may also contain nonprime attributes with partial and indirect dependence on the candidate keys. These undesirable associations are removed from a relation by normalisation can be defined as a step-by-step reversible process for transforming an unnormalised relation into relations of progressively simpler structures. Since the process is reversible, no information is lost during the transformation. Codd has defined three stages of normalisation known as the first

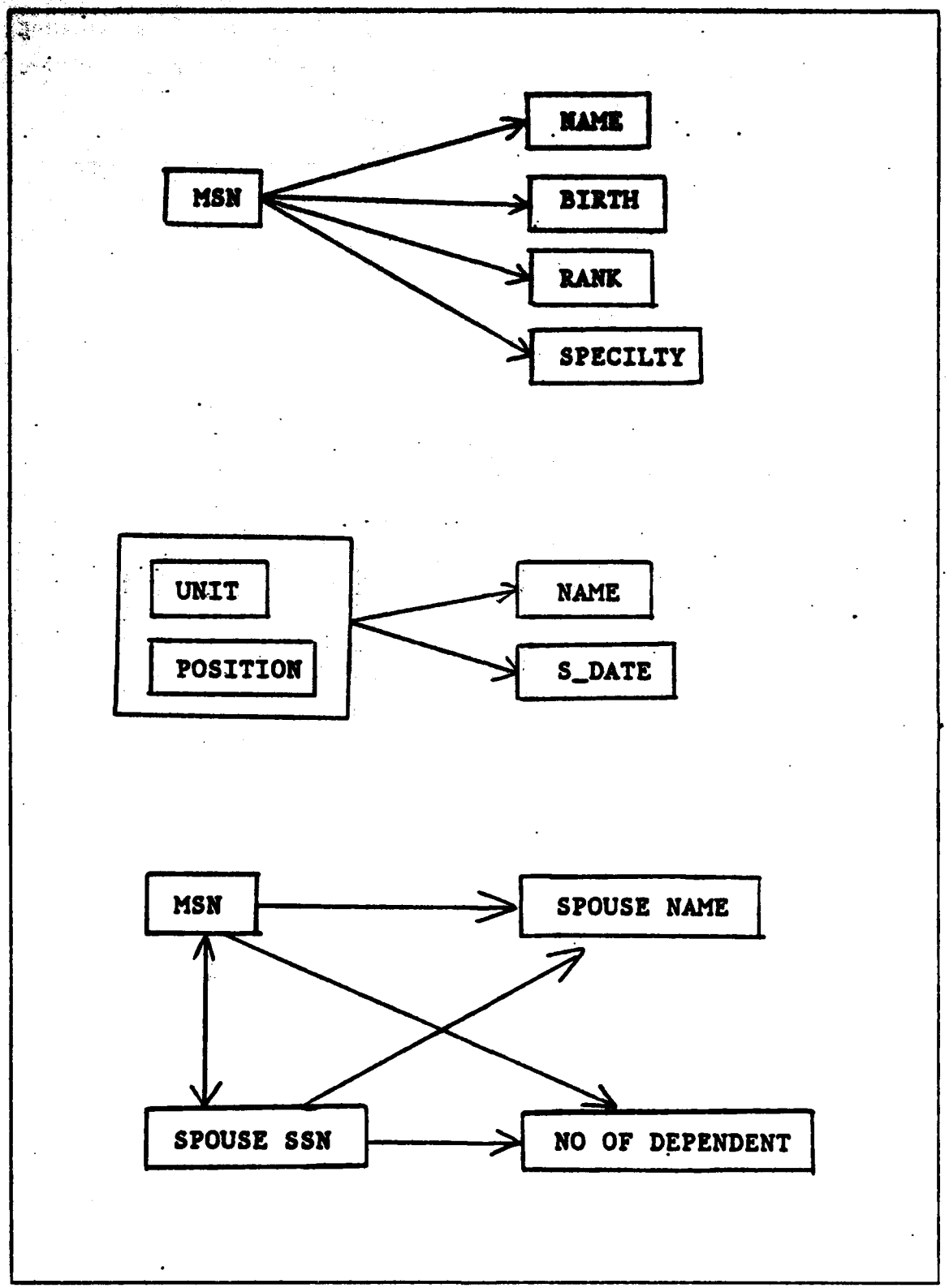


Figure 4.5 Functional dependencies in relation PERSON,ASSIGNMENT,FAMILY.

(1NF), second (2NF), and third (3NF) normal forms corresponding to the three types of undesirable association discussed above, namely, the elemination of the repeating groups, partial dependence and indirect dependence. The levels of normalisation are shown in Figure 4.6. [Ref. 8: p.135]

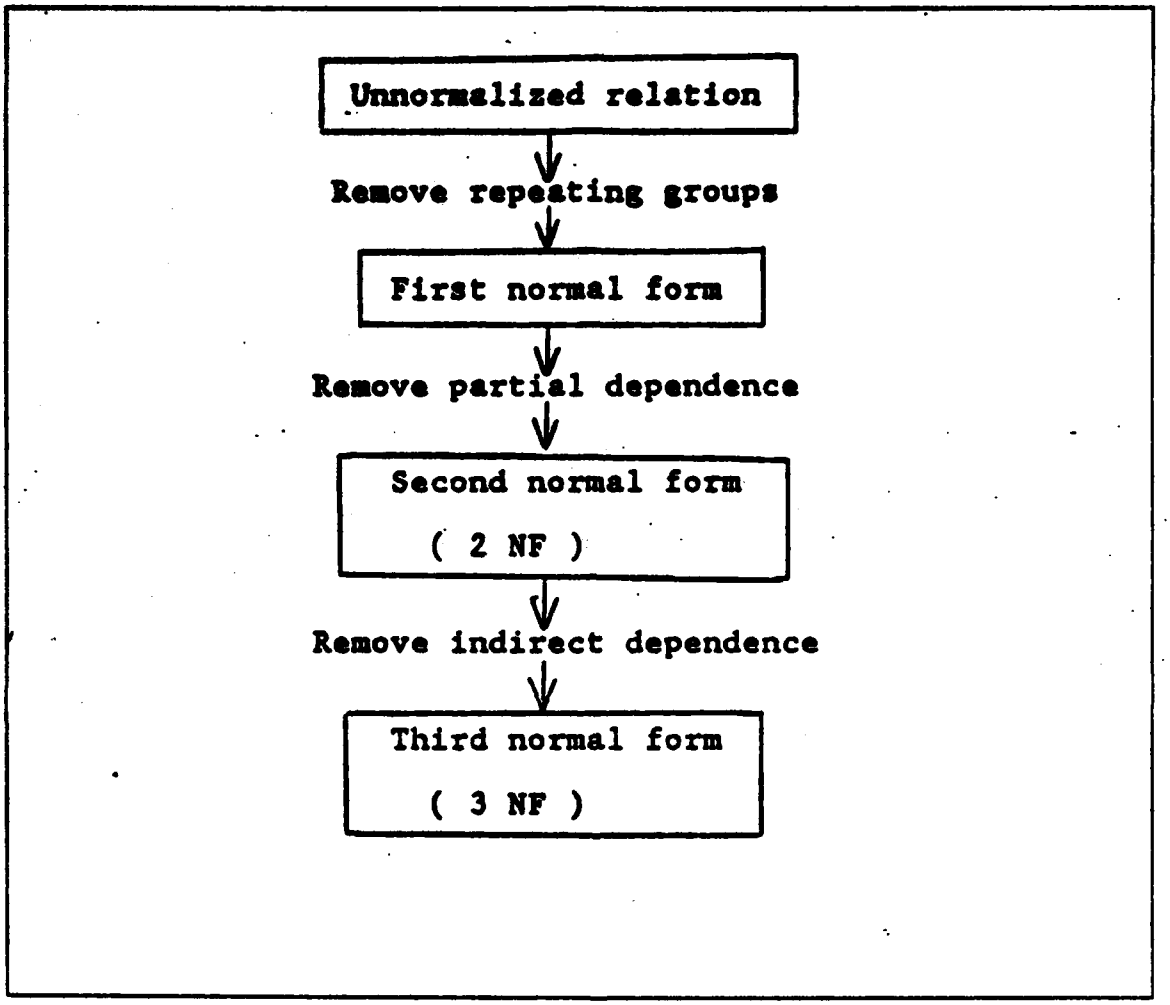


Figure 4.6 Three levels of normalisation.

a. First normal form

First normal form is the starting point, that is, all relations are in first normal form. An unnormalised relation is transformed into 1NF by splitting the relation into two, one for the repeating groups and the other for the rest. Consider the relation CAREER.

CAREER(MSN,RANK,NAME,UNIT,JOB,LOCATION,S_DATE,E_DATE)

1111	Capt.	Jae Park	222SQ	A	K1	83.4.1	83.10.21
			223SQ	B	K2	83.10.22	84.11.18
2222	Lt/col.	Sam Kim	225SQ	C	K3	84.8.4	85.7.15
			226SQ	D	K4	85.7.16	86.8.5

Figure 4.7 An unnormalised relation with repeating group.

In order to select a specific person who is most proper for a specific position (or job), his/her career is very important. In that case, unnormalisation can be made easily as it is shown in Figure 4.7. This clearly unnormalised relation, since it includes the repeating groups of item code. This relation is transformed into first normal form by splitting it into two relations PERSON and JOB as it is shown in Figure 4.8.

b. Second normal form

As it is shown in Figure 4.6, partial dependence is removed from 1NF in 2NF. The second normal form is formally defined in terms of what is called functional dependence. A normalised relation is said to be in the second normal form if all its nonprime attributes are fully dependent on each candidate key, in other words, if non prime attributes do not show any partial dependence on the candidate keys. In Figure 4.7, the attribute JOB is fully functional dependence on the collection of domain (MSN + UNIT). But the LOCATION is independent of the MSN and is therefore only partially dependent on the key (MSN + UNIT). Finally the 2NF is shown in Figure 4.9.

c. Third normal form

As it is shown in Figure 4.6, indirect dependence is removed from 2NF in 3NF. A normalised relation is said to be in third normal form if all its nonprime attributes are fully functionally and directly dependent on each candidate key. To demonstrate transitive dependence,

PERSON(MSN, RANK, NAME)

1111	Capt.	Jae Park
2222	Lt/col.	Sam Kim

R1(MSN, UNIT, JOB, LOCATION, S_DATE, E_DATE)

1111	222SQ	A	K1	83.4.1	83.10.21
1111	223SQ	B	K2	83.10.22	84.11.18
2222	225SQ	C	K3	84.8.4	85.7.15
2222	226SQ	D	K4	85.7.16	86.8.5

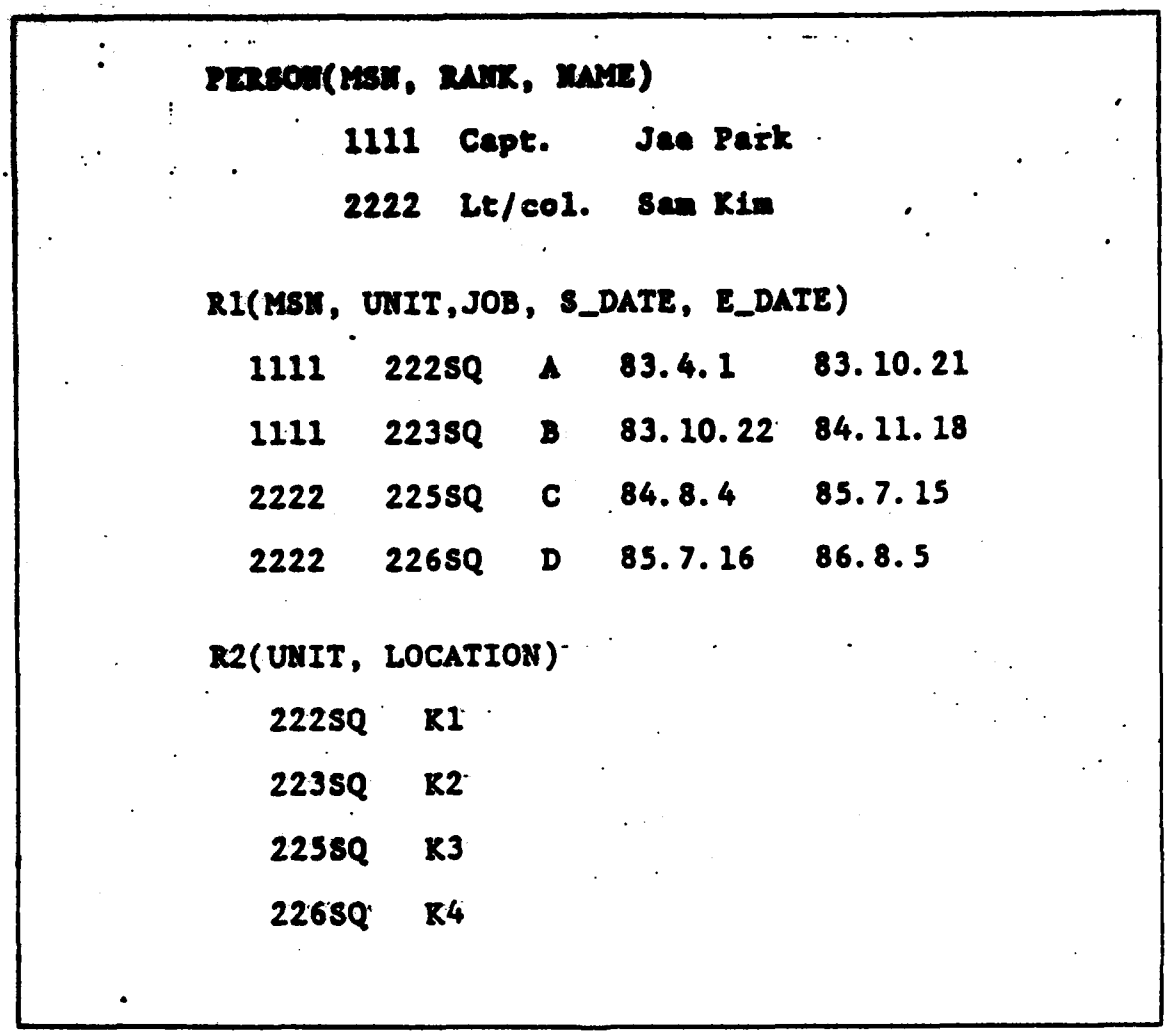
Figure 4.8 Relations in 1NF of Figure 4.7.

Let us consider relation MEMBER (Figure 4.10)containing Squadron, Pilot_name(P_NAME), Quantity of squadron(QOS), Rank. If SQ is the candidate key then this relation is not in 3NF, since the nonprime attributes RANK is not directly dependent on SQ. They are dependent on P_NAME, which is dependent on SQ. We convert this into third normal form by splitting as shown in Figure 4.11.

Transitive dependence causes update problems similar to those caused by partial dependence. Therefore all relations must be expressed in 3NF. On optimal third normal form is defined as the minimum number of relations that can express the original unnormalised relation.

d. Fourth and fifth normal form

Fourth and fifth normal forms deal with multivalued facts. A multivalued fact may correspond to a many to many relationship or to a many-to-one relationship. Under fourth normal form, a record type should not contain two or more independent multivalued facts about an entity. In addition, the record must satisfy third normal form. Fifth normal form deals with cases where information can be reconstructed from



PERSON(MSN, RANK, NAME)

1111	Capt.	Jae Park
2222	Lt/col.	Sam Kim

R1(MSN, UNIT,JOB, S_DATE, E_DATE)

1111	222SQ	A	83.4.1	83.10.21
1111	223SQ	B	83.10.22	84.11.18
2222	225SQ	C	84.8.4	85.7.15
2222	226SQ	D	85.7.16	86.8.5

R2(UNIT, LOCATION)

222SQ	K1
223SQ	K2
225SQ	K3
226SQ	K4

Figure 4.9 Relations in 2NF of Figure 4.7.

smaller pieces of information which can be maintained with less redundancy. Roughly speaking, we may say that a record type is in fifth normal form when its information content can not be reconstructed from several smaller record types, i.e., from record types each having fewer fields than the original record. Fifth normal form does not differ from fourth normal form unless there exists a symmetric constraint. One advantage of fifth normal form is that certain redundancies can be eliminated.

We discuss two additional normal forms very briefly here, in order to give some idea as to how normalisation research is continuing.

MEMBER(SQ, P_NAME, QOS, RANK)			
222SQ	Jae Park	50	Capt.
223SQ	Sam Kim	60	Maj.
224SQ	Won Hong	70	Maj.

Figure 4.10 A relation showing transitive dependence.

MEMBER1(SQ, P_NAME, QOS)		
222SQ	Jae Park	50
223SQ	Sam Kim	60
224SQ	Won Hong	70

MEMBER2(SQ, P_NAME, RANK)		
222SQ	Jae Park	Capt.
223SQ	Sam Kim	Maj
224SQ	Won Hong	Maj.

Figure 4.11 The relations of Figure 4.10 in 3NF.

E. SCHEMA DESIGN

1. View of the schema

A relational database is specified by a relational schema which consists of one or more relational subschemas. A relational subschema is a listing of a relation name and its corresponding attributes. Figure 4.12 represents an example of a relational schema.

These views are then integrated to form an enterprise description which describes the entire conceptual schema. This description is used mainly for communication between the users and the schema designers. For each entity type identified, a description of the entity type is produced and the associated data classes

```
PERSON(MSN, Name, Rank, Birth)
    key : MSN
REWARD/PUNISHMENT(MSN, A_rank, Type, Data, Reason)
    key : MSN + Type + Date
MT(MSN, C_name, S_date, E_date, Period, Grade)
    KEY : MSN + C_name + S_date + E_date
  * MT:MILITARY TRAINING
```

Figure 4.12 An example of a relational schema.

identified. The description names the entity type, defines what it represents, and lists its associated attributes.

2. **Identifying constraints**

In order to complete the enterprise description step, identify constraints on the, attributes, entity types, and relationship types. It seems better to state all constraints explicitly rather than as inherent constraints. To help identify constraints, the following questions are posed:

(1) what is the domain of values for each attribute?

(2) What are the known functional dependencies between attributes of each entity type? (it is discussed in detail in previous section)

(3) What are the keys for each entity type? (it is discussed in detail Chapter III)

(4) What are the predicate constraints to be placed upon the data?

It is difficult to arrive at a set of constraints that represents the application and its consistent and feasible, because some forms of the constraints are difficult understand and are prone to misunderstandings and errors. Figure 4.13 shows domains and attribute/domain correspondence based on Figure 4.12.

F. TRANSACTION CONSIDERATION

The final phase of the enterprise description step identifies the transaction-processing requirements of the organization with respect to the enterprise description.

Domain Name	Format and Meaning
MSN	positive integer less than 10000
Name	char(20);full names of person
Rank	char(20);person's rank
Birth	numeric YYMMDD
A_rank	char(20);person's rank when he/she was Rewarded or punished
C_name	char(40);military training course name
Grade	value is 'A','B','C',or 'D'

Figure 4.13 An example of domains and attribute/domain correspondence for Fig 4.12.

All current and projected transactions are included. For each transaction, the designer identifies its nature (retrieval, update, delete, insert), its frequency, its origin (organizational area), and its purpose, together with the point(s) of schema it affects. Figure 4.14 shows a example of transactions. To help identify requirements for supporting transactions, the following questions are posed: What transactions are required by each organizational area? What kind of access is required by each transaction? What reports are needed? What entity types, attributes, and relationship types are involved in each transaction? etc.

Transaction: List all combat pilots who have some flying qualification, capability grade, and whose rank is captain.

Organizational area: Operational Department

Entity: PERSON(MSN,RANK,NAME)

COMBAT QUALITY/TRAINING(MSN,UNIT,FLY_Q,C_GRADE)

Relationship types: PERSON-COMBAT QUALITY/TRAINING

1. Retrieve PERSON entity(for MSN, RANK, NAME)
2. Retrieve all PERSON entities related to the COMBAT QUALITY/TRAINING via a PERSON-COMBAT QUALITY/TRAINING

Figure 4.14 A simple example of transaction.

V. SYSTEM ANALYSIS FOR RELATIONAL DESIGN

A. PROBLEMS AND USER'S REQUIREMENT

First of all,in order to design a database, we should interview with user and and decide output which they need. In case of the R.O.K Air Force, we should classify or break down pilots by their skill,flying hours, flying qualification etc. to select pilots who are best for specific jobs. This step consist of a high-level analysis of the function of an organization. Desired information of personnel managers or unit commanders might include:

1. List of all new commissioned officers in a specific year concerning scholarship, major, health condition, family condition, etc.
2. The number of cadets or candidates who should be commissioned in the next year or at a specified year for each source organization.
3. List all officers with each rank who graduated each military education course.
4. Selection of some officers for some positions.
5. Summary of an officer's career from a certain previous rank up to the current rank.
6. List certain officer's rewards or punishments.
7. Present an information list for promotion purposes for each rank and service branch, including career, result of fitnees reports, education, rewards and punishment, health condition, and the order of promotion recommendation,etc.
8. List all pilots who satisfy a certain level of flying hours, flying qualification and a certain type of aircraft.

All of information which may be required by personnel managers can not be desired, because different managers request different information. Personnel managers might need information for their job in addition to that described above. The purpose of requirement analysis is to:

(1) Gain familiarity with the area of the organization to be modeled

(2) Determine the information requirements of the organization without regard to constraints other than the way in which the organization does business.

(3) Represent these requirements via some formal modeling technique.

Some major personnel management aspects of the Republic of Korea Army are promotion selection, job assignment management, estimation of required personnel resources for education, Welfare-Morale management and payroll. We will consider and discuss only the data concerning the Personal Record, which is composed of basic information, education, career, etc. Record relationships and record structure will be discussed in detail in the next section.

Promotion Selection is managed by the Department of Personnel Management. To collect the general personnel data for promotion selection, 2 to 10 officers from each branch of the Army execute the routine job of manual data collection every year. Since it is a manual job, there exists the possibility of mistakes and it is impossible to provide the various kinds of data necessary to support promotion selection.

In the case of assignment management, it is difficult to examine the data for matching required personnel with available personnel resources for a specific job or position. Therefore, an officer may be assigned to an undesired unit or job because of a subjective decision made by the detailing officer. This has caused a great deal of personnel dissatisfaction with job assignments.

The Department of Personnel Management is responsible for promotion selection and job assignment management. The Central Financial Corp is responsible for payroll. Welfare management is the responsibility of the Welfare-Morale Corps. Because they maintain separate data, data integrity and accuracy is very low.

As was mentioned above, the numerous problems faced are :

(1) Wasting manpower and time due to manual processing causing work delays

(2) Ineffectiveness of work and non-integrated data processing due to individual software maintenance, and spending a relatively large amount of time on the maintenance effort and minor enhancements

(3) Lack of proper supporting system for decision-making

(4) Lack of data accuracy

Figure 5.1 shows the scope and objectives of the database that we have designed in this thesis.

B. MODELING

The essence of database design is the representation of record relationships. The relationships can be specified in a variety of ways. Data Structure Diagram (DSD also called Bachman Diagram) is a simple method used to represent overall record structures. The single or double arrow notation is used to express relationships between records (one-to-one, one-to-many, many-to-many relation ships). shows the relationships among records.

The relationships are identified intuitively. The design team considers potential relationships among records that have been defined. A relationship may exist among three, four or more records. At this point the design team must discriminate between theoretical and useful relationships. A theoretical relationship can exist logically, but

- Project : Personnel Management System.
- Problem : Individual and manual data processing.
- Objectives : To design and implement a prototype personnel management system.
  1. Providing proper data for decision making.
     a. Promotion selection.
     b. Assignment management.
     c. Management of personnel resources for education.
     d. Welfare & morale.
     e. Personnel supply.
  2. Reduction of manpower and time by integrated data maintenance and processing.
  3. Increase in data accuracy and work efficiency.

Figure 5.1 Statement of scope and objectives.

may never be needed in practice. Theoretical record relationships were discussed in Chapter III.

E. Record structure

In order to satisfy the user's requirement we will derive a number of records from personal data. Because of military security, we will prototype similar record model instead of displaying the whole personal record. From the information in the personal record, we can build a number of record by bundling a few items as fields in a relational model. In this thesis, we will generate the following records with the underlined field(s) as the key:

1) MAIN (*SN*, NAME, ORG_BRANCH, C_TYPE, BORN_DATE, BORN_PLACE)

SN : Service number

NAME : Name
ORG_BRANCH : Original branch
C_TYPE : Commissioned type
BORN_DATE : Born date
BORN_PLACE

2) M_EDUCAT (CNAME, CLASS_SIZE, START_DATE, END_DATE, SNAME, CLASS_MEAN)
CNAME : Course name
CLASS_SIZE : Class size
START_DATE : Start date
END_DATE : End date
S_NAME : School name
CLASS_MEAN : Class mean points

3) EDUCATMN (SN, CNAME, E_GRADE,MEAN) ---- Intersection record
SN : Service number
CNAME : Course name
E_GRADE : evaluation grade
MEAN : Personal mean points

4) RANK (RANKS, P_ORDER, TDATE)
RANKS : Rank
P_ORDER : Personnel order
T_DATE : Date

5) PROMOTE (SN, P_ORDER) ---- Intersection record
SN : Service number
P_ORDER : Personnel order

6) AWARDPUN (KIND, P_ORDER, T_DATE)
KIND : Kind of award and punishment
P_ORDER : Personnel order
T_DATE : Date

7) A_P_MN (SN, P_ORDER) ---- Intersection record
SN : Service number
P_ORDER : Personnel order

8) A_P_P (KIND, POINT)
KIND : Kind of award and punishment
POINT : Points given by award and punishment

9) EXPERT (SN, EXPERTITLE)
SN : Service number
EXPERTITLE : Expert title

10) P_EVAL (SN, GRADE, TDATE)
SN : Service number
GRADE : Performance evaluation
T_DATE : Date

11) CAREERS (SN, SE_NO, START_DATE, END_DATE, P_ORDER) ----- Intersection record
SN : Service number
SE_NO : Serial number
STAR_TDATE : Start date
END_DATE : End date
P_ORDER : Personnel order

12) UNIT (SE_NO, DUTY_TITLE, UNIT)
SE_NO : Serial number
DUTY_TITLE : Duty title
UNIT : Unit

Precise information about each field is in Data Dictionary in Section C. of this chapter.

2. Record relationship diagram

As was mentioned before, the fields in the MAIN record are fixed items that never need to be changed. Therefore, as we can see in Figure 5.2, all other records are

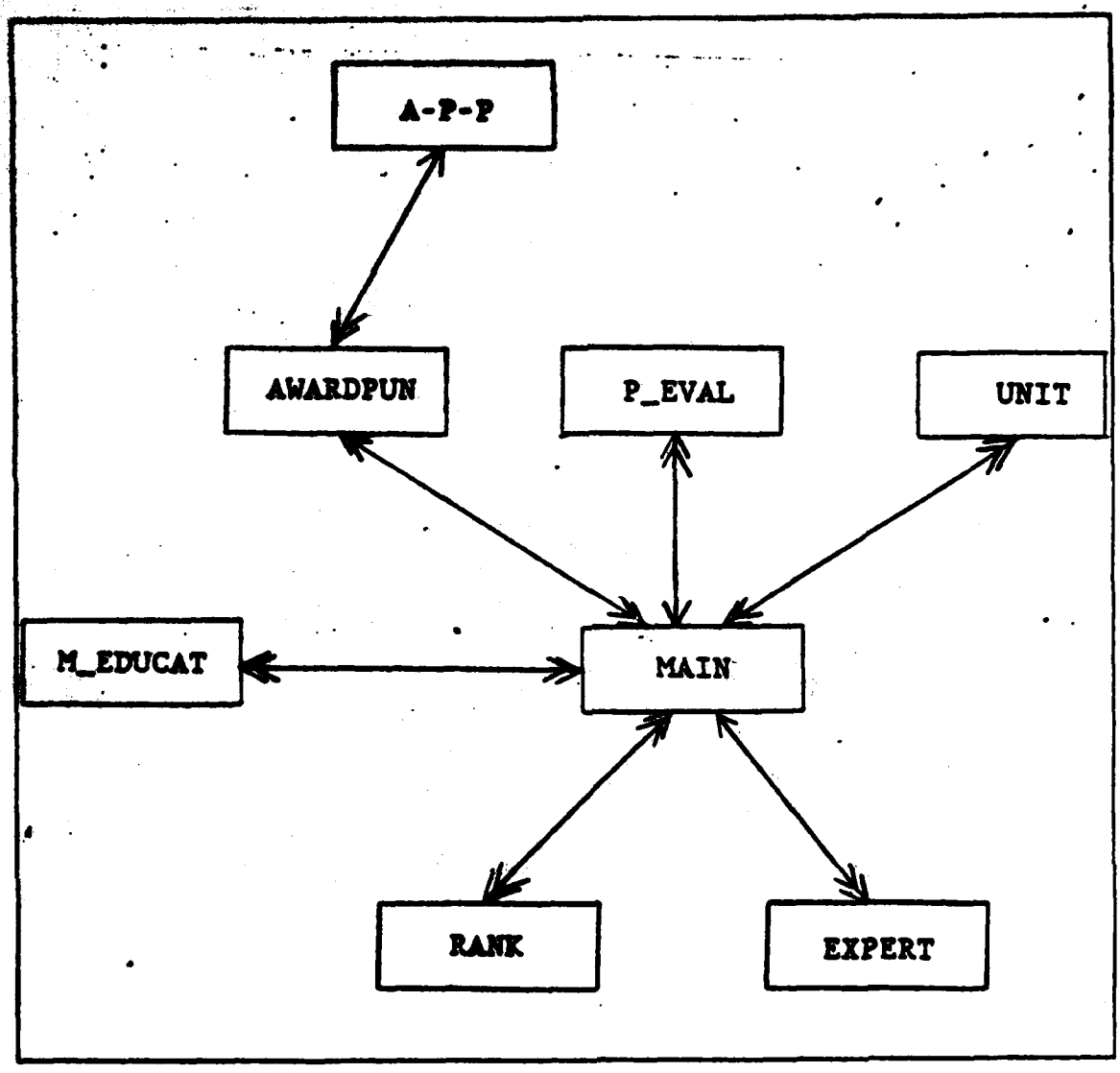


Figure 5.2 General view of record relationship diagram.

centrally related to the MAIN record. Each record has its own individual purpose, for example, AWARDPUN record shows information about award or punishment which a certain person was given ,M_EDUCAT record shows training and education that a certain person has taken. The reason that we show record structure in the previous section is to make it easier for the user to understand what record is needed for what purpose. Relationships in Figure 5.2 should be reorganized by intersection record to divide complex network record relationships. Reducing a complex network to a simple relation is discussed in Chapter III. Figure 5.3 shows the final record relationship design with intersection record.

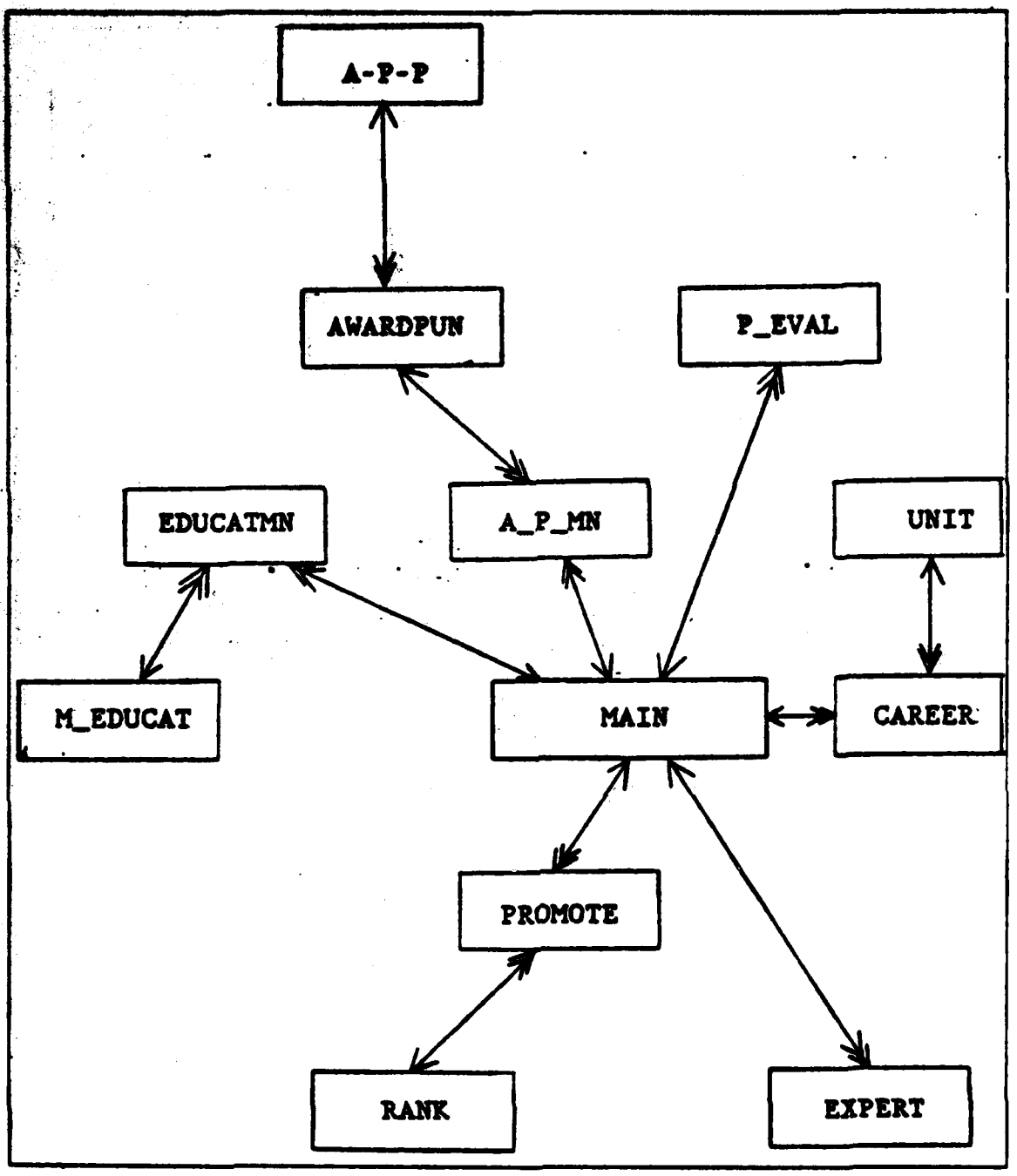


Figure 5.3 Record relationship diagram with intersection record.

C. DATA DICTIONARY

A data dictionary could be defined as a collection of correct information about the words and terms used by an organization to describe its data. The term data

dictionary is used to indicate a collection of information, that is, a set of files or a database. The term is also used to dscribe the mechanism for storing data in the fields and databases, that is, a system or a collection of computer programs. Data dictionaries as files or databases contain data which are physically stored on magnetic storage media, most often magnetic disk. Data dictionary systems as a collection of computer programs perform the functions of storing, retrieving, and quite often manipulating and passing data on to other systems, like the DBMS.

The six major steps in building a DDS(Data Dictionary System) that will respond to the enterprise's need for having complete control over their data resource follow: [Ref. 11: p.56]

(1) Establishing data naming and definition standards/conventions: This includes standardization of data elements, data items, and data definition (DD) naming conventions, program naming conventions, and job name naming conventions, at minimum.

(2) Establishing standard abbreviations and acronyms: This includes standardization of abbreviations and acronyms, as well as of establishing the rule to define a term the first time it is being used in programs, documentation,reports,etc.

(3) Identifying and defining "base data" data elements: This includes the identification and definition of "product data" and the "division or department data" of the enterprise, both of which are essential for the company's existence.

(4) Identifying and defining codes: This includes identification and definition of code types of data elements.

(5) Identifying, defining, and standardizing input, update, and validation procedures: This includes identification, definition, and standardization of I/O, update and validation.

(6) Identifying and defining data characteristics: This includes the identification and definition of the characteristics of data.

Management of a database is usually a complex process. It requires the database administrator to keep track of all the database and user view definitions as well as their use. Data dictionaries have been developed to aid the database administrator in this task. The generation of the data dictionary which documents functions, data bases, allowable values, formats, and their interrelationship should be initiated at this point.

The Data Dictionary for this thesis is as follow:

Field name: sn

Format: character

Width: 8

Allowable value: less than 100,000,000

Description: military service number of person.

for example, 77-15-3376, 73-05-5253.

Field name: name
Format: character
Width: 25
Allowable value: last name, first name middle name
Description: name of person

Field name: org_branch
Format: character
Width: 20
Allowable value: infantry, engineer,etc.
Description: original branch that a person was
assigned when he/she was commissioned.
sometimes person work in another branch
for some period of time.

Field name: c_type
Format: character
Width: 10
Allowable value: ROTC, KMA(Korean Military Academy),etc.
Description: commisioned type that is identified by the
school or education before commision.

Field name: born_date
Format: character
Width: 8
Allowable value: YYMMDD
Description: date of birth

Field name: born_place
Format: character
Width: 15
Allowable value: city of Seoul, Chunnam do,..
Description: birth place. a special city, a city under the

direct control of the government or do(province)

Field name: c_name
Format: character
Width: 20
Allowable value: infantry OBC, engineer OAC,etc.
Description: military course name. a list of the types
of training required for a certain job or
rank.

Fieldname: e_grade
Format: character
Width: 15
Allowable value: outstanding, middle,etc.
Description: evaluation grade which is given at the
end of every training or education course.

Field name: mean
Format: numeric
Width: 5
Allowable value: less than 100
Description: personnel average value

Field name: class_size
Format: numeric
Width: 4
Allowable value:less than 500
Description: number of students in class.
class size is needed when a person's position
in class should be given for evaluation.

Field name: start_date

Format: character
Width:8
Allowable value: YYMMDD
Description: date when he/she starts a certain assigned job or education.

Field name: end_date
Format: character
Width:8
Allowable value: YYMMDD
Description: date when he/she finish a certain assigned job or education course. start date and end date is needed to know time and duration of person's assigned job or education course.

Field name: sname
Format: character
Width: 8
Allowable value: Army infantry school,Army engineer school,etc.
Description: school name. many military training or education courses are provided by many different schools.

Field name: class_mean
Format: numeric
Width: 5
Allowable value: less than 100
Description: class mean grade

Field name: rank
Format: character
Width: 20

Allowable value:2LT,1LT,Capt,etc.
Description: person's rank

Field name: p_order
Format: character
Width:20
Allowable value: 77-33 army, 78-13 army,etc.
Description: personnel order for education, assignment,
promotion etc. p-order has many different
type of serial number for each type of order.

Field name: t_date
Format: character
Width: 20
Allowable value: YYMMDD
Description: date

Field name:kind
Format:character
Width:20
Allowable value:staff of chief awarding, corps commander
awarding,etc.
Description: kind of award or punishment

Field name: point
Format:numeric
Width:4
Allowable value: greater than -5 and less than 5
Description: different point is given depend on the type of
award or punishment.

Field name: expertitle

Format: character
Width: 20
Allowable value: CPA, civil engineer,etc.
Description: some people have special qualification.

Field name: grade
Format: character
Width: 2
Allowable value: aa,ab,ba,cb,etc.
Description: this field is composed of two grades, the first
grade is given by commander and the second
grade is given by vice-commander by the level
of job accomplishment.

Field name: dutytitle
Format: character
Width: 20
Allowable value: platoon leader,company commander,etc.
Description: duty title. some job or assigned position
is required for promotion. duty title is
needed for job assignment or promotion selection.

Field name: unit
Format: character
Width: 25
Allowable value: 55x 227r 2bn 5co 2pl,48x 105r 70bn,etc.
Description: unit is composed of division(x),
regiment(r),battalion(bn),company(co),platoon(pn).

Field name: se_no
Format: numeric
Width: 7
Allowable value: 9293001,7354023,etc.

Description: serial number. unique number is given to every unit.

VI. DATA BASE IMPLEMENTATION

A. INTRODUCTION

The introduction of a data base as the central reservoir of data affects the user organization in a number of ways. For example, it changes the organization's attitude to data requirements and management, it creates new authorities and it brings in new skills. It also enforces greater coordination between the various user departments and demands stricter adherence to standards. A good implementation scheme should include adequate provision to tackle these problems, in addition to having plans for system developments and scheduling of resources. Much of this would be planned and controlled by the DBA, on whose ability will largely append the success of the venture - provided that the right DBS is selected in the first place. In this chapter we will discuss these issues: relational implementation and data base administration.

B. RELATIONAL IMPLEMENTATION OVERVIEW

In relational systems, data relatibility is provided by the ability to construct new relations from existing relations by the use of the relational operators. The relational operators may require access paths to contain or represent the derived relations. The access paths may exist in the system, or they may be constructed by the system as required. For instance, a join can be implemented as a separate file, as a set of pointers, or by storing the definition of the join and generating it as needed. However, no matter how it is implemented, the user is not aware of the exact representation and does not really care what it is.

Since a user is not explicitly aware of the access paths in a relational system, this may lead to the misconception that relational systems do not provide access paths. This is far from true. As in a hierarchical or a network system, a relational system may need specific nature. The existence, and maintenance of these access paths may be hidden from the user. Nevertheless, they exist, and their implementation is one of the hardest design problems in a relational system.

Relational systems can be differentiated according to how they represent derived relations via access paths. If they only store the definition of a derived relation, then the access paths corresponding to the physical implementation of the derived relation are destroyed after every query. In this case, content addressability may be sufficient

to construct the access paths as required. For example, a join can be constructed using content addressability by correlating tuple identifiers from the inverted files, for the two relations, according to the join condition. On the other hand, the system can retain and maintain the access paths corresponding to the definition of a derived relation. In this case, the maintenance of these access paths can become very involved.

There are currently many commercial DBMS products that claim to be relational. Some are more relational in name than in actuality. The DBMS should model data as tables, and it should support SELECT, PROJECT, and unrestricted JOIN operations. A system that supports restricted JOIN operations falls in a gray area. Some people would call the system a relational system in spite of this limitation. Others would call it a tabular system.

Relational DBMS can be divided into three groups. One group is based on the data language SQL, one on the data language QUEL, and one group contains system falling into neither of these categories. Three major SQL-based DBMS products are SQL/DS, System R, and ORACLE. System R is a research system developed by IBM for the study of relational technology. System R has been used in a prototype mode by several major industrial concerns. SQL/DS is a commercial version of System R.

ORACLE was developed for operation on Digital Equipment Corporation PDP minicomputers. Since its organization, ORACLE has been converted to operate on IBM mainframes as well. ORACLE's user interface is based on SQUEL II, an earlier version of SQL. According to RSI, ORACLE will soon be compatible with the current version of SQL.

QUEL (QUEery Language) is a data language like SQL. QUEL is based on tuple relation calculus. QUEL is nonprocedural and allows the user to process data without concern for physical data structures. The data base product INGRES is based on QUEL. INGRES operates on Digital Equipment PDP hardware and runs under the UNIX operating system. IDM 500 is also based on QUEL.

There are many other relational DBMS. There is even a microcomputer relational product: dBASE II, which is needed by Ashton-Tate. dBASE II operates on CP/M-based micros. dBASE II is an example of a relational DBMS that restricts join operations. The join columns must be indexed. [Ref. 7: pp.437,438]

C. IMPLEMENTATION USING DBASE III+

As we mentioned before, we use DBASE III+ to show the sample application program for prototyping in this thesis. The Korean military personnel management system has been implemented using dBASE III+ relational DBMS in appendix. As a word processor allows one to manipulate characters, words, sentences and pages to create a document that fits one's needs, dBASE III+ allows one to work with fields, records, and files to manage data in just the desired manner.

We will provide basic operations to implement the data base using dBASEIII+ in this section.

1. CREATE

First of all, in order to create a file, CREATE command is used as follows:

```
. CREATE PARK
    Field Name  Type       Width   Dec
---------------------------------------
1   RANKS       Character     20
2   P_ORDER     Character     20
3   TDATE       Character      8
```

2. USE

A file called PARK has now been created on the data base by 1. To select a file to work with, we should use USE command:

```
. USE PARK
```

3. APPEND

To add the information to the file, use the APPEND command. APPEND lets the user move the cursor to any field and enter or change the information.

```
. APPEND

RANKS       2nd lieutenant
P_ORDER     77-33 army
TDATE       03/28/77

RANKS       1st lieutenant
P_ORDER     78-33 army
TDATE       04/01/78
```

4. LIST

When we create a file or add information to a certain file, we need to verify the information and data structure. In this case, we use LIST command to list a data file's contents on the screen.

```
. LIST
Record#  RANKS                P_ORDER              TDATE
      1  2nd lieutenant       77-33 army           03/28/77
      2  1st lieutenant       78-33 army           04/01/78

. LIST FOR RANKS = "2nd lieutenant"
Record#  RANKS                P_ORDER              TDATE
      1  2nd lieutenant       77-33 army           03/28/77


. LIST STRUCTURE
Structure for database: C:PARK.dbf
Number of data records:        1
Date of last update    : 01/09/87
Field  Field Name  Type        Width     Dec
    1  RANKS       Character      20
    2  P_ORDER     Character      20
    3  TDATE       Character       8
** Total **                     49
```

[redacted]

[redacted] If the contents of this file is changed, the EDIT command can be used. [redacted] allows full screen operation.

```
. LIST
RECORD#   RANKS                P_ORDER          TDATE
     1    2nd lieutenant       77-33 army       03/28/77
     2    1st lieutenant       78-33 army       04/01/78

. EDIT 2

RANKS      1st lieutenant
P_ORDER    78-33 army
TDATE      04/01/78

. list
RECORD#   RANKS                P_ORDER          TDATE
     1    2nd lieutenant       77-33 army       03/28/77
     2    major                78-33 army       04/01/78
```

6. DELETE

To delete records from a delete file, we use DELETE command.

```
. LIST
Record#  RANKS                P_ORDER          TDATE
      1  2nd lieutenant       77-33 army       03/28/77
      2  major                78-33 army       04/01/78

. DELETE RECORD 2
      1 record deleted
. DISPLAY
Record#  RANKS                P_ORDER          TDATE
      2 *major                78-33 army       04/01/78

. RECALL
      1 record recalled
. DISPLAY
Record#  RANKS                P_ORDER          TDATE
      2  major                78-33 army       04/01/78

.
. DELETE RECORD 2
      1 record deleted
. DISPLAY
Record#  RANKS                P_ORDER          TDATE
      2 *major                78-33 army       04/01/78

. PACK
      1 record copied
. LIST
Record#  RANKS                P_ORDER          TDATE
      1  2nd lieutenant       77-33 army       03/28/77
```

7. SELECT

To use more than one data file, dBASE III+ reserves two areas of memory for data file. If it is necessary to use another data file at the same time, SELECT command must be used.

```
. SELECT 1
. USE PARK
. SELECT 2
. USE PROMOTE INDEX PROMOTE
. JOIN WITH PARK TO TEMPFILE FOR P_ORDER =A->P_ORDER FIELDS SN,A->RANKS
     2 records joined
. USE TEMPFILE
. LIST
Record#  SN       RANKS
      1  20001    2nd lieutenant
      2  21554    2nd lieutenant
```

8. INDEX

We can use INDEX command to sort a data file in a certain order. If a specific item, not the whole list, is wanted, then the FIND command can be used to display on the screen.

```
. INDEX ON P_ORDER TO KIM
  100% indexed              2 Records indexed
. USE PARK INDEX KIM
. LIST
Record#  RANKS                P_ORDER              TDATE
      1  2nd lieutenant        77-33 army           03/28/77
      2  1st lieutenant        78-33 army           04/01/78

. FIND 77-33 army
. DISPLAY
Record#  RANKS                P_ORDER              TDATE
      1  2nd lieutenant        77-33 army           03/28/77
```

B. DATA BASE ADMINISTRATOR

In a conventional system, files belong to the relevant user departments. It is they who are responsible for the accuracy, consistency and up-to-dateness of data in the file, although regular maintenance on their behalf is normally carried out by the data processing staff. In a data base where all company data are centrally held, no single user department can be responsible for it. Instead this responsibility is exercised by the data base administrator on behalf of the whole company with a view to preserving the interest of both current and future users. In addition, the DBA is also responsible for creating, expanding and improving the data base and for providing user facilities. For a small data base, the function of the DBA can be performed by an individual as a part-time job, but for a large data base, the function can require the full-time services of a team. To be effective, the DBA should represent a senior position with sufficient authority to arbitrate disputes between the user departments on data base usage, and to impose decisions in case of deadlocks. He should also be accepted as the final authority in all matters relating to the management of the data base.

The function of the DBA should be include the following activities: creation of the data base, performance optimization, data protection, specification and enforcement of standards, and coordination and the provision of the user facilities. David defines the responsibility of DBA as follow:

(1) DBA Data Activity Management Responsibilities
- provide data base standards
- establish data owner ship, retrieval, and modification rights
- create and disseminate recovery procedure
- inform and train users
- enforce data activity policy
- publish and maintain documentation.

[Ref. 7: p.540]

(2) DBA Database Structure Management Responsibilities
- design the schema(s)
- provide design expertise
- control redundancy
- maintain configuration control of change requests
- schedule and run configuration control meeting
- implement schema changes

- [illegible] user documentation
- [illegible] DBA documentation.

[Ref. 7: p.540]

(2) . DBA Responsibilities for Database System Management

- generate data base system performance reports
- investigate user performance complaints
- analyze reports and complaints
- tune the data base system
- tune communications software and operating system to data base (when possible)
- evaluate and implement new features.

[Ref. 7: p.540]

VII. SAMPLE PROGRAM PROCESSING

Personnel management system (PMS) is a menu driven program organized along functional lines. Each major function in the system corresponds to a section of the User's manual and selection from the main menu. It is important to read the information on the feature you plan to use before you use it. This tells you exactly what information you need and where to find it.

There is also a chapter on the DRIVER (or main menu) section of PMS. This is very important to read before you start using PMS as it describes the actions required if you get stuck in the middle of a program and cannot get out or if the system crashes.

This manual is indicated for use with the IBM PC(AT). You have been provided 3 diskettes: dBASE III+(I), dBASE III+(II), and PMS. You must "boot up" your computer system (which should be equipped with a hard disk) and then do the following:

1. Copy above diskettes into the hard disk.
2. Type "dbase" (no quotes needed), and then strike the return key. Once the system is loaded, you will see the "."(dot) prompt on the screen.
3. Type "set default to c"(no quotes needed) and then the return key, the "."will appear again. Now you are ready to use the personnel management database system(PMS).
4. At this point, type "do driver"(no quotes needed) followed by a carriage return. After a short wait, the PMS main menu will appear and you can start enjoying PMS.
5. You have to follow the command messages which appear on the screen. The various options of the main function of this system are specified in the first message (main menu). Enter the main menu-letter which you want. This will be followed by another message menu on the screen. Select the menu-letter which you want, and follow the command message on the screen.
6. After having done what you wanted, you can return to the previous menu by selecting menu option "x".

A. DRIVER

The driver program has three functions: First, it will ask what kind of service you require and then call that function your task. It will continue until you finished. This is a very straight-forward menu driven selection process and does not require any special information to use. Second, much more complicated part of the DRIVER, deals with reconstruction of the Data Base if the system crashes in the middle of a session. The DRIVER always stores a copy of every user interaction in a file called CRASH.TXT. This file is deleted at the end of every normally terminated run. However, if the run were to abort abnormally this file would allow you to recreate every step and check the contents of the Data Base for accuracy. Finally, if you change the data in all system programs, the data is used continuously after changing.

The DRIVER program presents the main menu on the screen(Figure 7.1). If you are unfamiliar with the 7 options presented in the main menu, refer to the preliminary information section so that you can refresh yourself on what each option is all about. Select an option from 1 to 7. Upon option selection please refer to the respective section you have just selected.

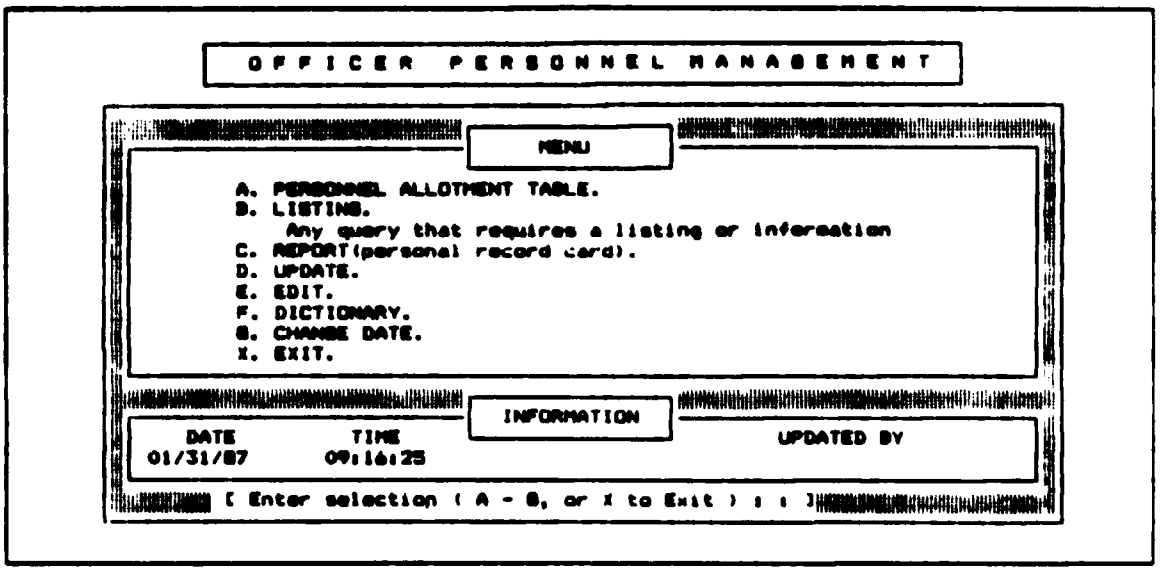


Figure 7.1 Officer personnel management.

B. TABLE

The Table function allows you to obtain the number of officers for each rank. You can reach this section by entering an "A" at the main PMS menu, and then the following screen(Figure 7.2) will appear.

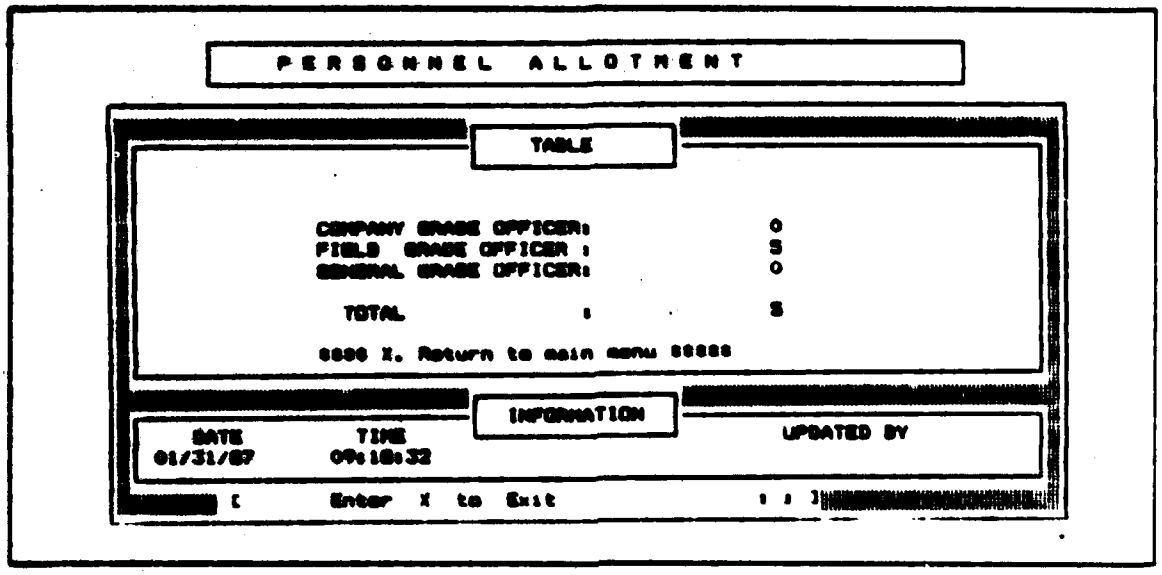


Figure 7.2 Personnel allotment.

C. LISTING

The listing function allows you to obtain listing of information on officers. You can reach this section by entering an "B" at main PMS menu. It is designed to cycle back to the main LISTING Menu until all of your queries are answered. It will then return you to the main PMS menu.The available listings are shown the following screen(Figure 7.3). Please pay close attention to the section on what information you need to access each report.

1. Select option "A" to list all officers

As the title suggests this section will provide a list of all officers in the data file. The output is shown below (Figure 7.4).

2. Select option "B" to list all new officers for assignment

This section provides a list of all new officers for assignment. The following screen(Figure 7.5) will appear. In order to use this section you will need the personnel order of the new officer exactly as it is shown in the database. A complete list of the

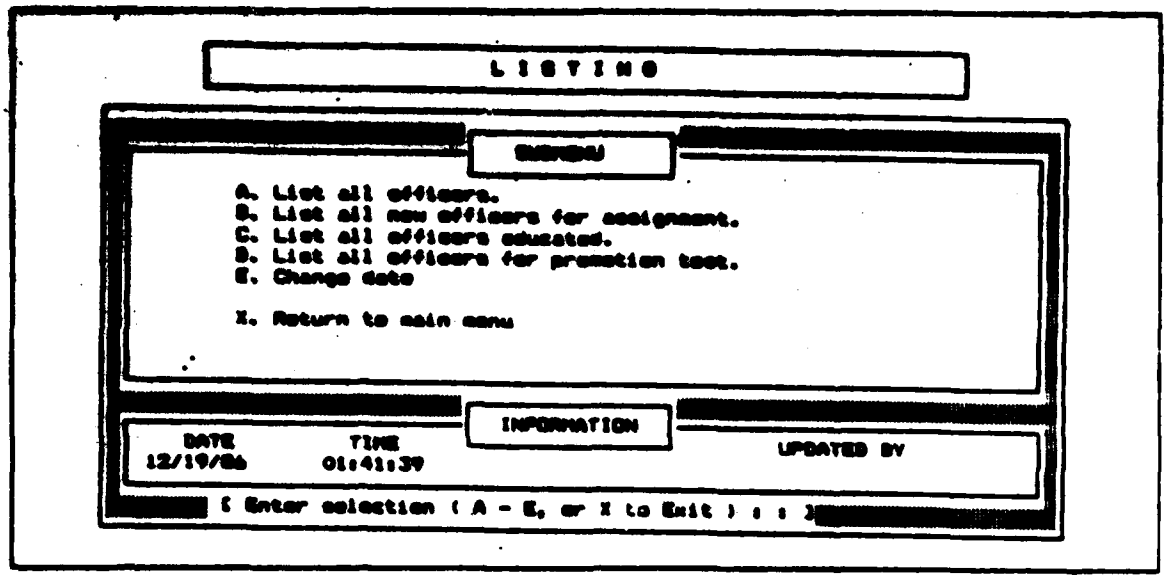


Figure 7.3 Submenu of listing.

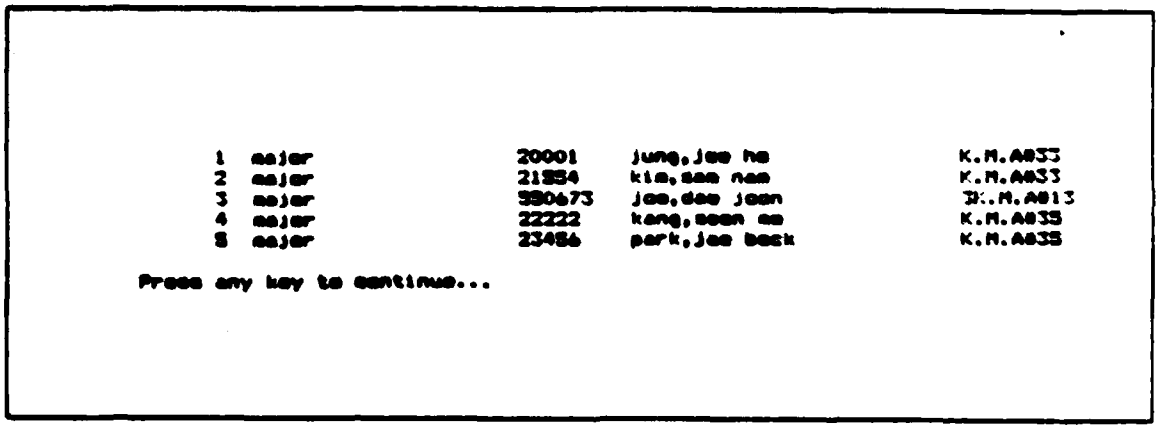


Figure 7.4 List all officers.

personnel order can be found in the Appendix of this thesis. You must enter the personnel order exactly as shown or the system will not be able to match the correct personnel order and will return an error message. The outputs are shown below (Figure 7.6).

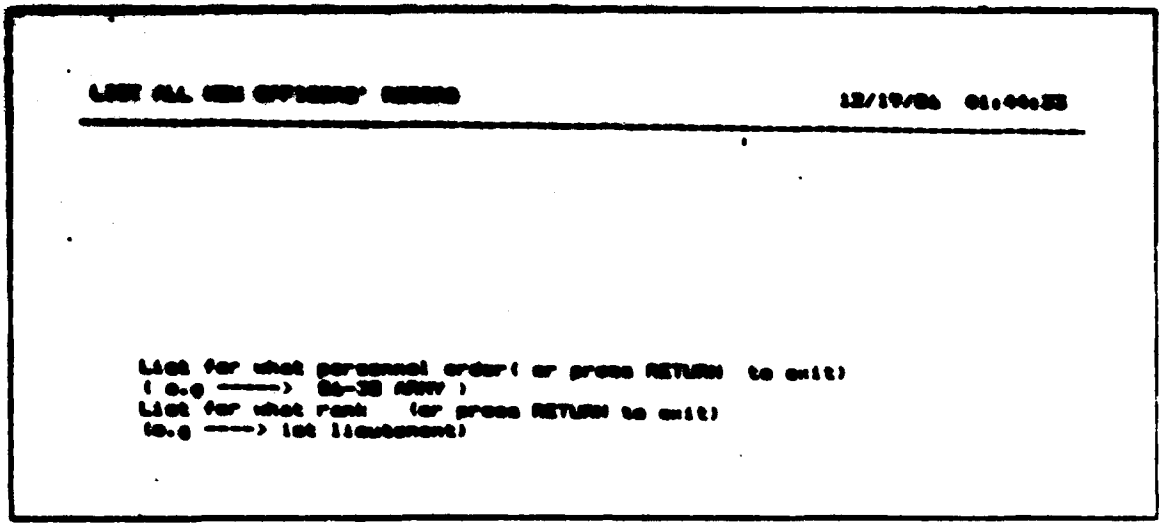


Figure 7.5 Query to find out commissioned officers.

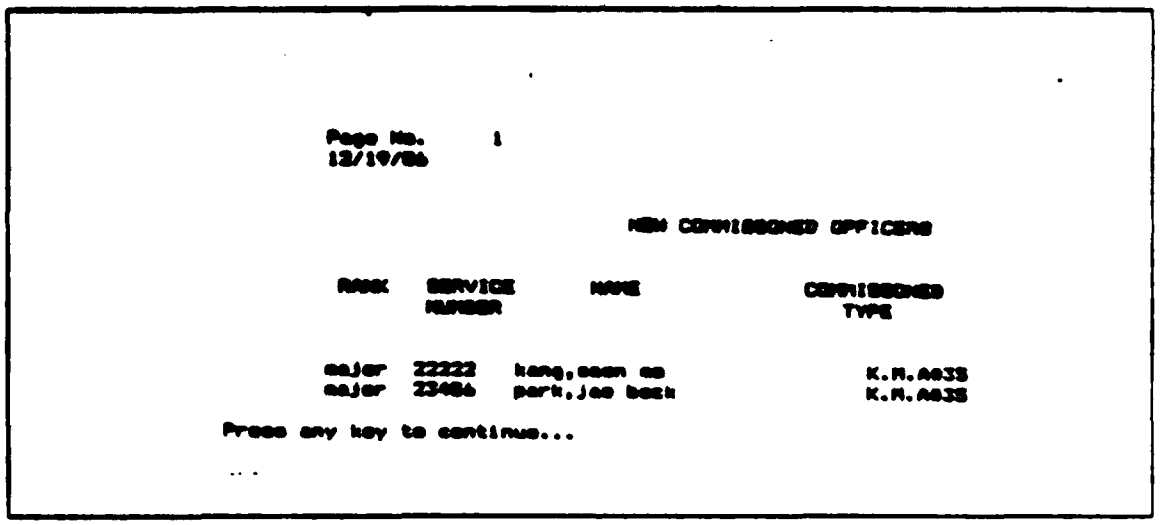


Figure 7.6 List new commissioned officers.

3. Select option "C" to list all educated officers

It will provided a list of the officers educated in the military school. The following screen(Figure 7.7) will appear. In order to use this section you will need to enter the military education course name in the Appendix exactly. The outputs are shown below(Figure 7.8).

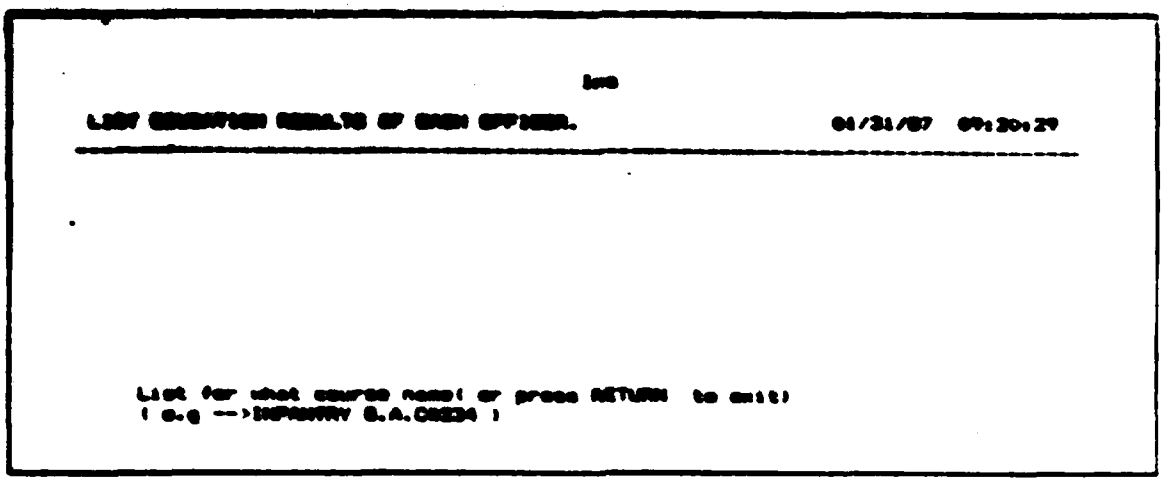


Figure 7.7 Query education results of each officer.

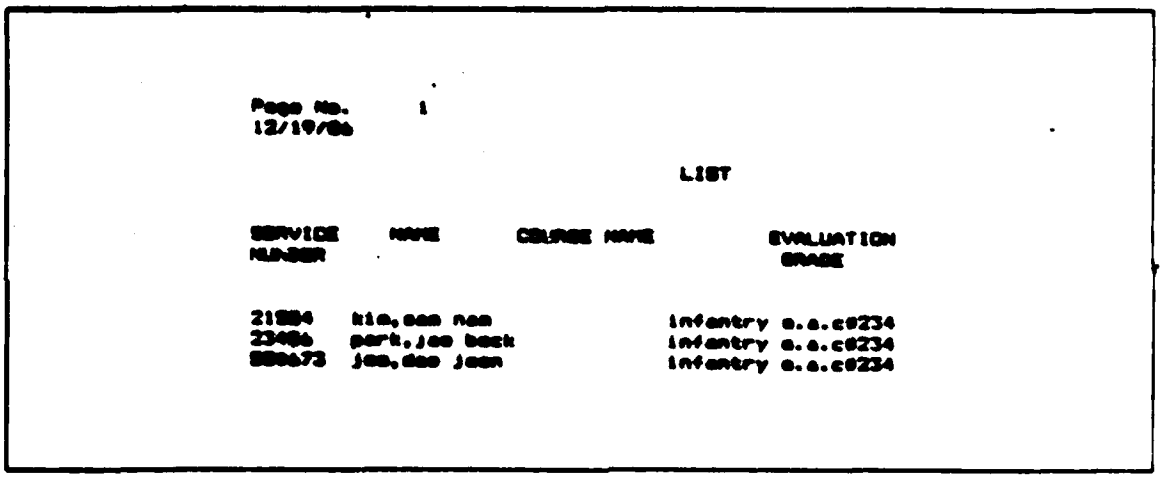


Figure 7.8 List education results of each officer.

4. Select option "D" to list all officers for promotion test

This will provide a list of all officers for the promotion test. The following screen(Figure 7.9) will appear. In order to use this section you will need the promotion year and rank (in the Appendix) exactly. The outputs are shown below(Figure 7.10).

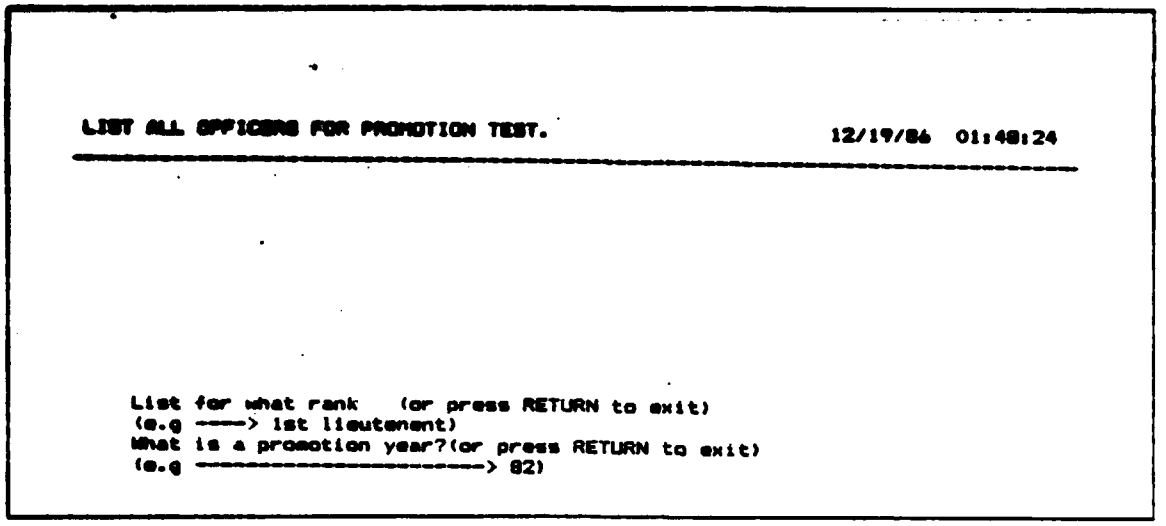


Figure 7.9 Query to find out all officers for promotion test.

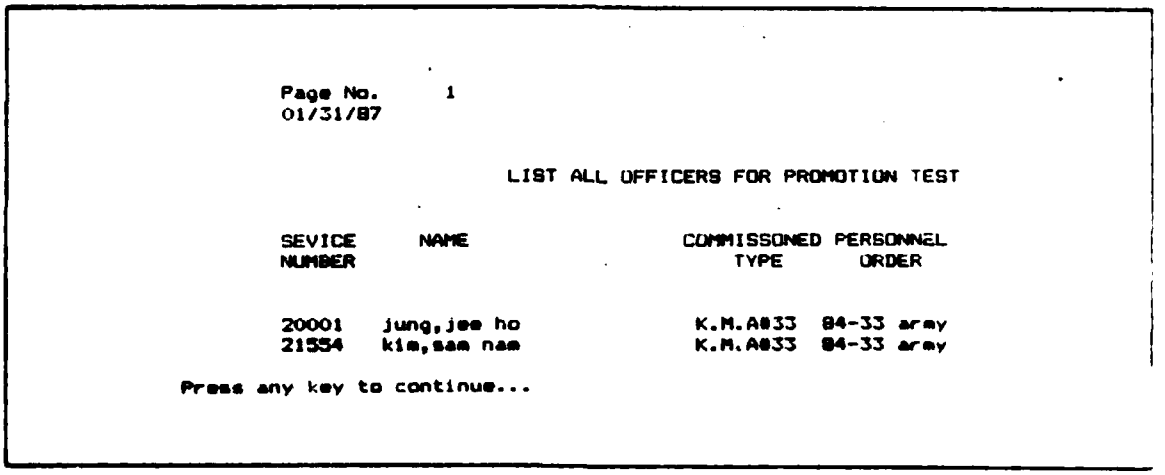


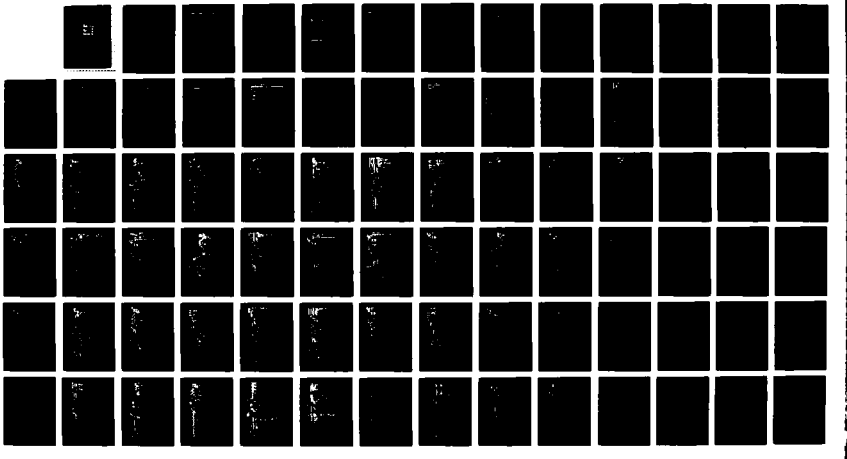
Figure 7.10 List all officers for promotion test.

D. REPORT (PERSONNEL RECORD CARD)

The REPORT function allows you to obtain detailed information of each officer's history. You can reach this section by entering a "C" at the main PMS menu. You will remain in the REPORTS.PRG untill all of your queries are answered then return you to the main PMS menu. The following screen figure appear. In order to use this section you will need the service numbers shown in the data base. A complete list of the service numbers Appendix of this thesis. The outputs are shown below Figure

AD-A181 663 APPLICATION OF A DATABASE SYSTEM FOR KOREAN MILITARY PERSONNEL MANAGEMENT(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA S N KIM ET AL. MAR 87 2/2

UNCLASSIFIED F/G 5/1 NL



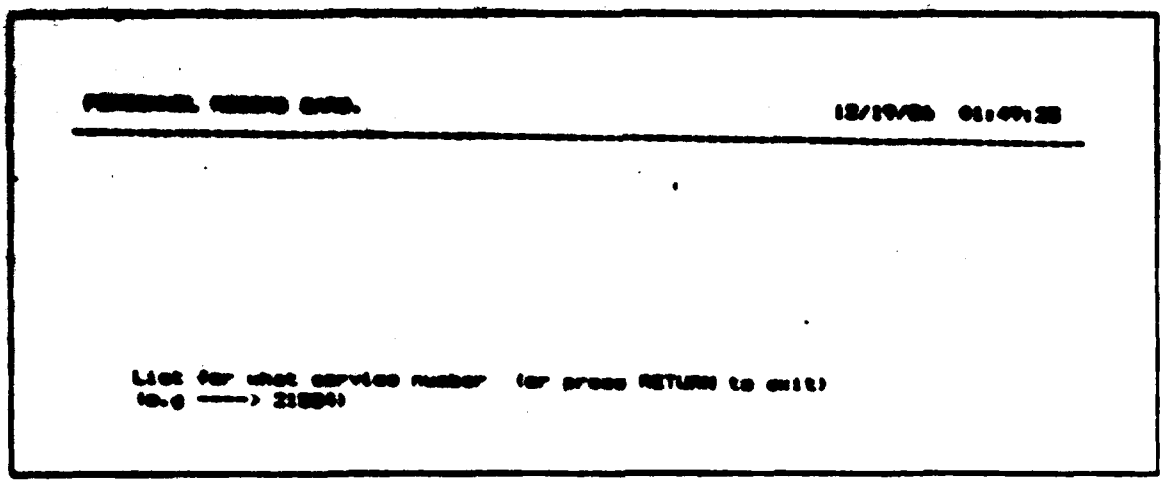


Figure 7.11 Query to find out personnel record card.

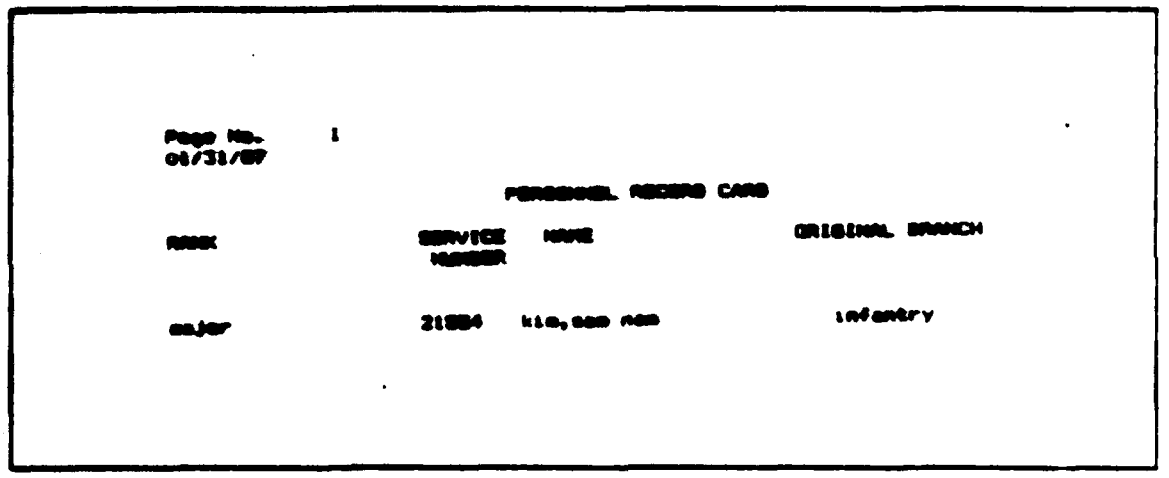


Figure 7.12 Personnel record card.

E. UPDATE AND EDIT

The UPDATE and EDIT function allow you to change fields in any file. It also changes any other files that were affected by your changes. It is very important that you are exact with your changes because the impact of a mistake could have wide reaching implications. You can reach this section by entering "D" and "E" for adding and changing at the main PMS menu. The update and edit program are almost the same. Therefore, This will be an explaination of the EDIT function only. The following sub-menu(Figure 7.13) will appear on the screen.

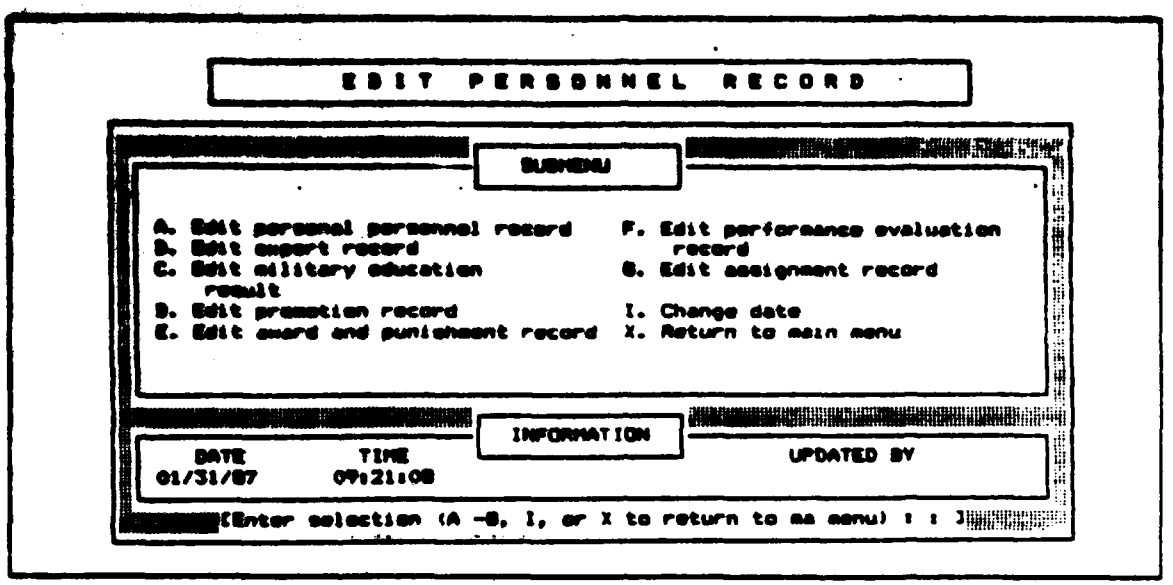


Figure 7.13 Submenu for changing personnel records.

1. Select option "A" to change personnel records

This will allow you to change any fields in the MAIN.DBF file. You must be very careful when changing the service number because the other files will be changed to this menu value as well as the MAIN file. The following screen(Figure 7.14) will appear. You must know the service number. Failure to do so may result in the wrong information. After entering the service number, the following screen(Figure 7.15) will appear, and then you may enter new data or change data.

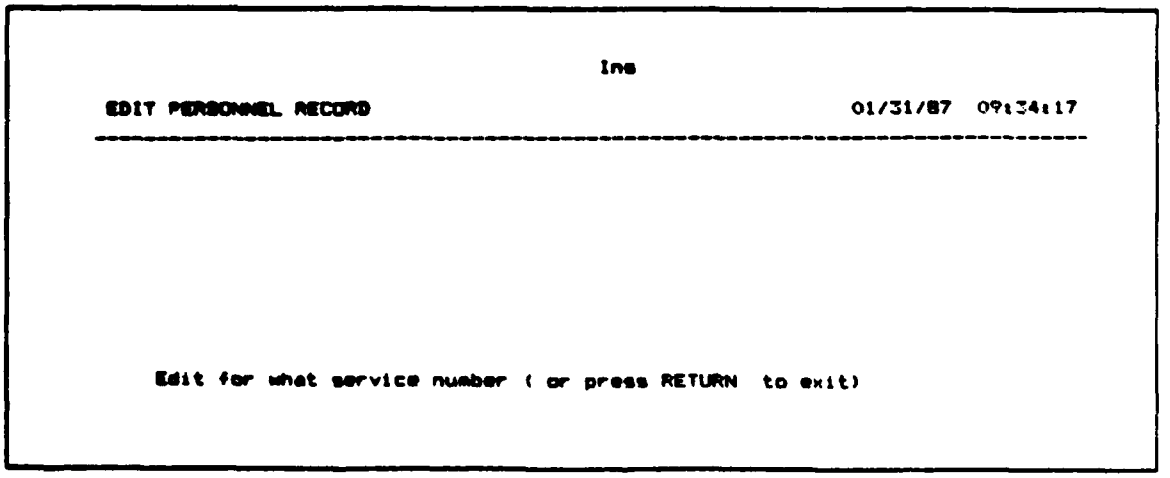


Figure 7.14 Query to change personnel records.

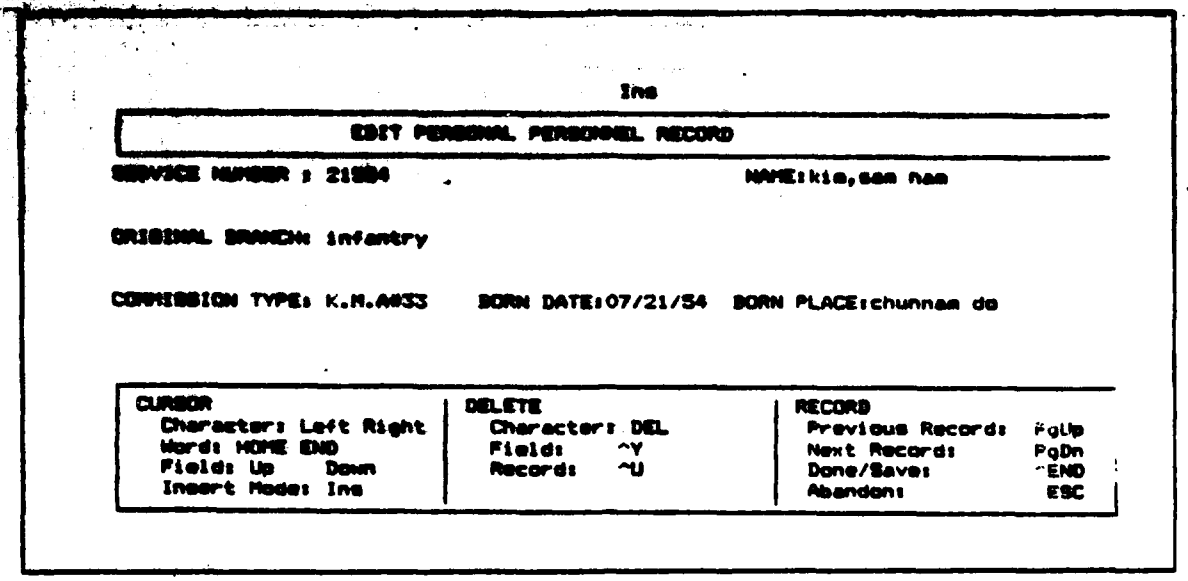


Figure 7.15 Screen for changing personnel records.

2. Selection option "B" to change expert records

This will allow you to change any files in the EXPERT.DBF file. The following screen(Figure 7.16) will appear. You must know the service number and expert title, failure to do so may result in the wrong information. After entering the service number and expert title, the screen will be displayed as follows(Figure 7.17), and then you may enter or change data.

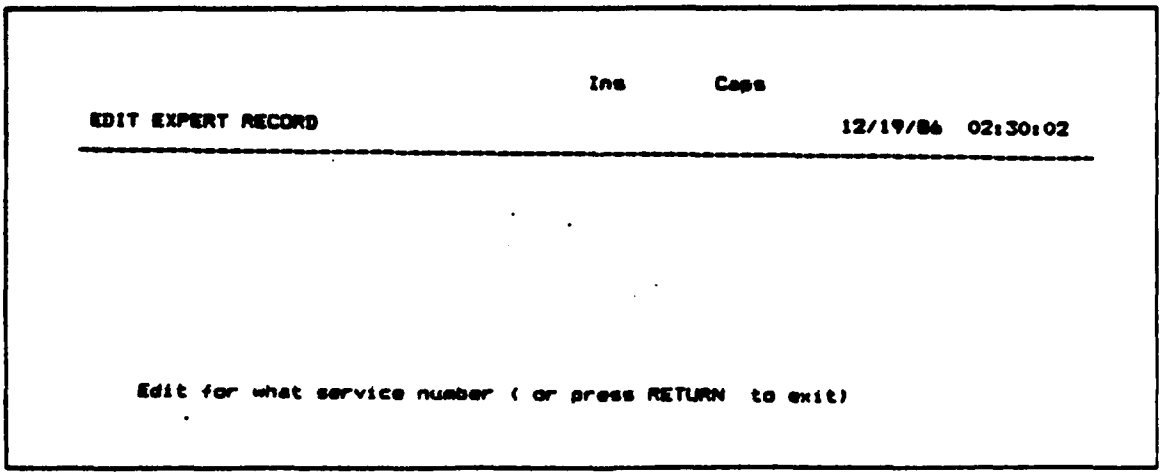


Figure 7.16 Query to change expert records.

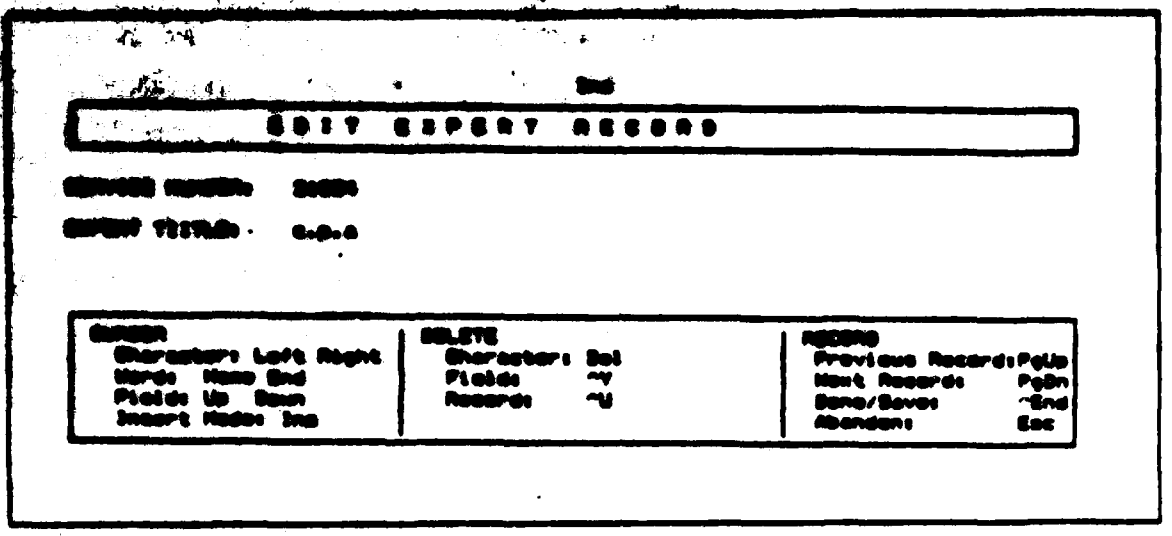


Figure 7.17 Screen for changing expert records.

3. Select option "C" to edit military education results

This allows you to change military education results. you will see the following sub-menu screen(Figure 7.18).

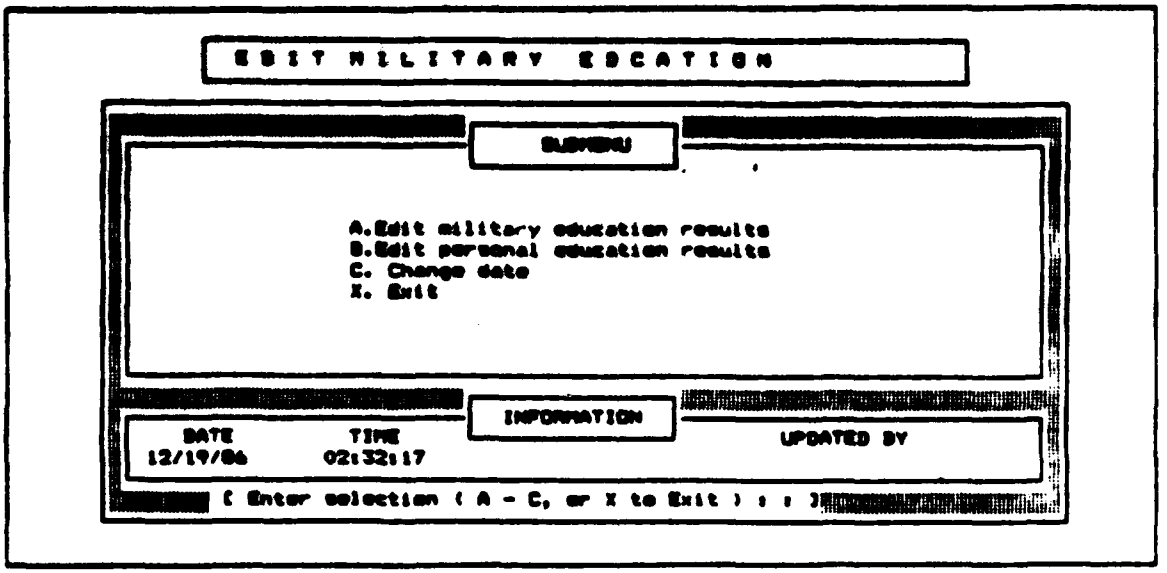


Figure 7.18 Submenu for changing military education results..

If you select option "A" from the military education sub-menu, you will see the following screen(Figure 7.19). This will allow you to change any files in the M_EDUCAT.DBF file. You must know the military education course name(in Appendix) exactly. After entering the course name, the following screen(Figure 7.20) will appear, and then enter or change data.

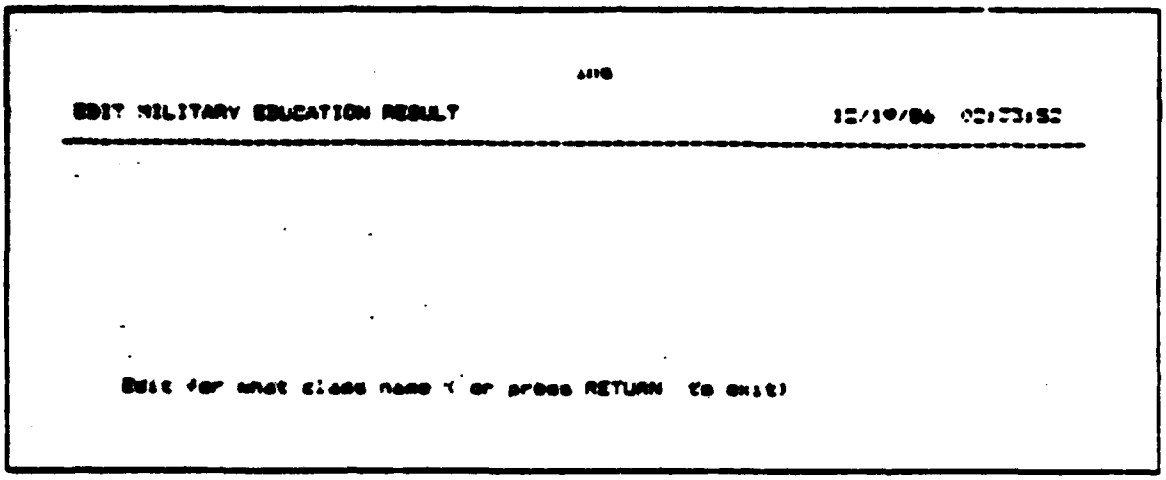


Figure 7.19 Query to change military education results.

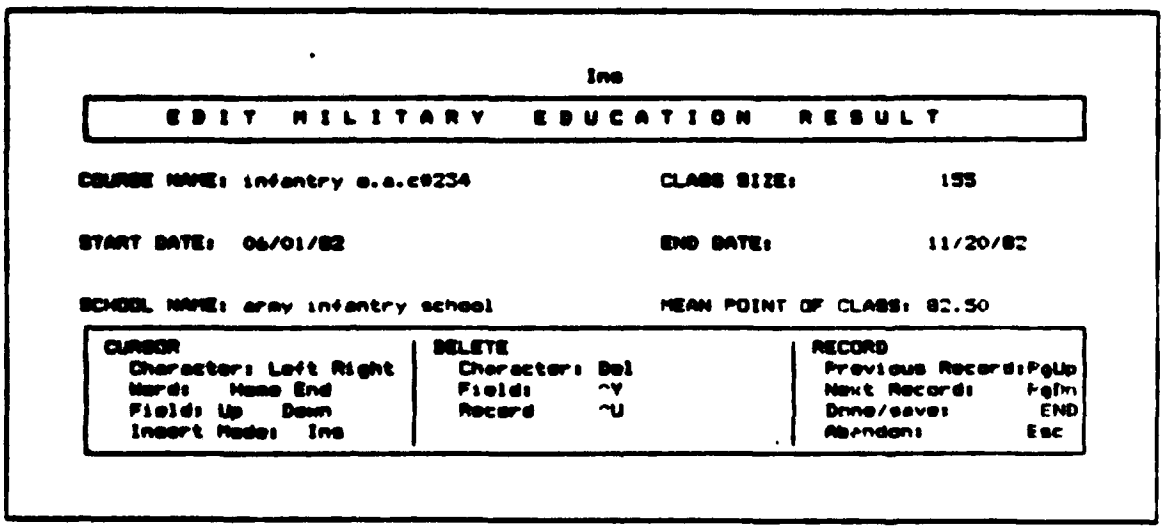


Figure 7.20 Screen for changing military education results.

If you select option "B" from the military education sub-menu, you will see the following screen(Figure 7.21). This will allow you to add and change any fields in the EDUCATMN.DBF. you must know the military education course name and service number (in Appendix) exactly. After entering the course name and service number, the following screen(Figure 7.22) will appear on the screen, and then you may enter or change data.

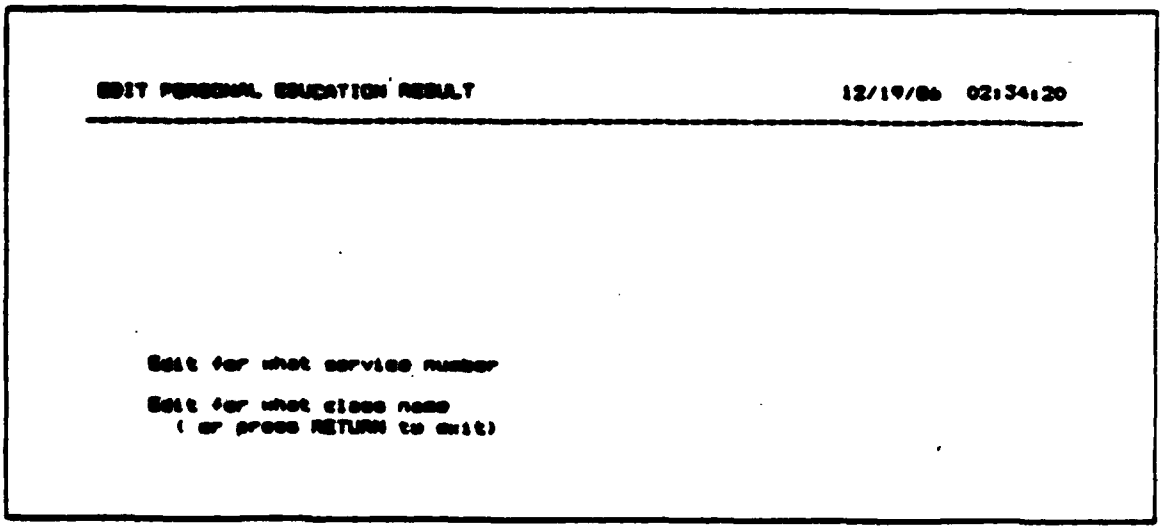


Figure 7.21 Queries to change personal education result.

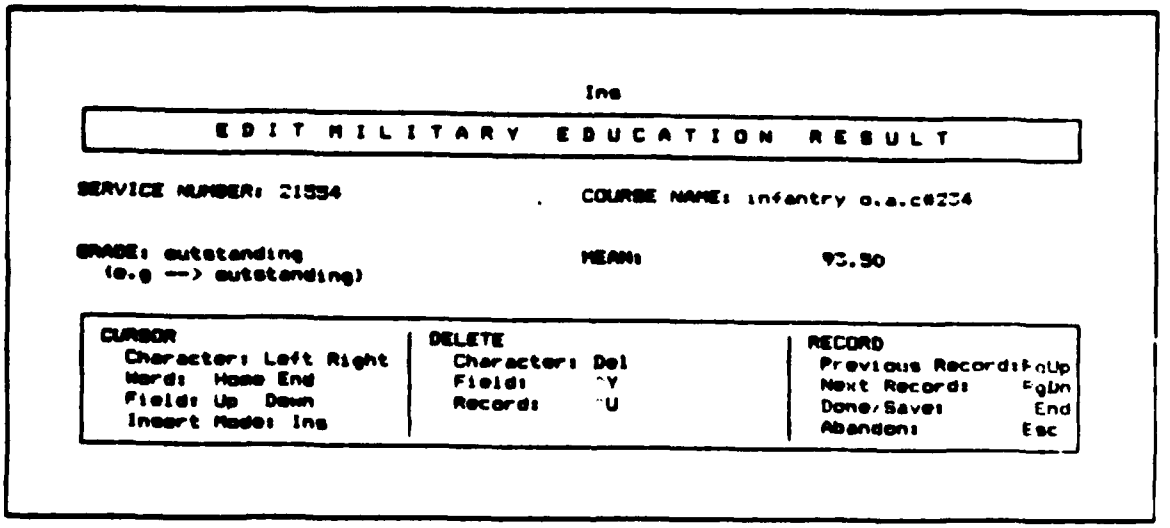


Figure 7.22 Screen for changing personal education results.

4. Select option "B" to change the promotion records
you will see the following sub-menu screen(Figure 7.23).

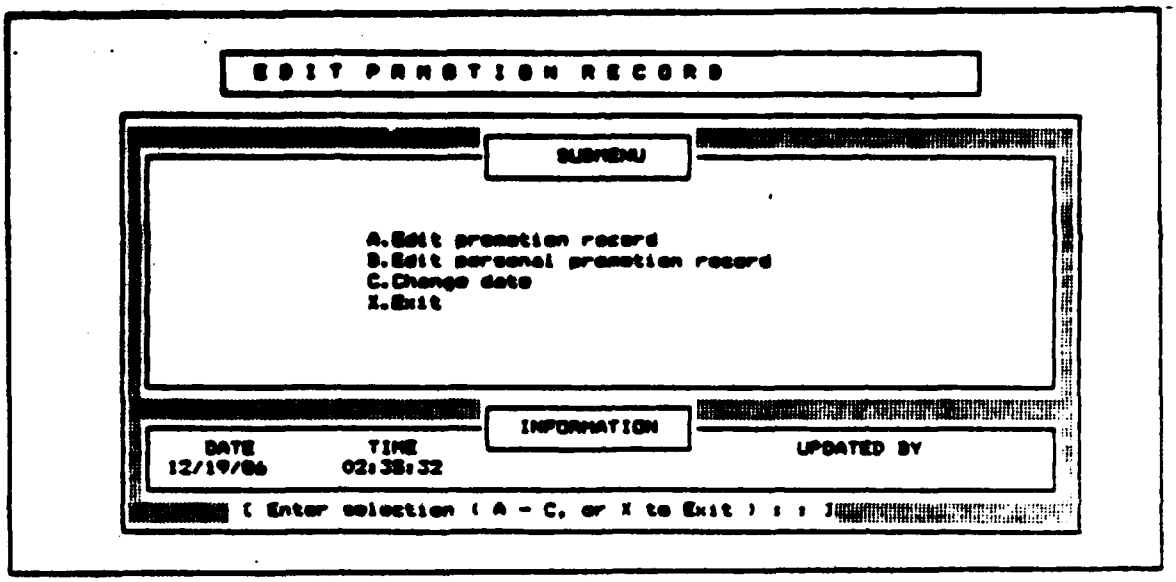


Figure 7.23 Submenu for changing promotion records.

If you select option "A" from the above sub-menu, you will see the following screen(Figure 7.24). This will allow you to add and change any fields in the RANK.DBF file You must know the promotion order(in Appendix) exactly. After entering the promotion order, the following screen(Figure 7.25) will appear, and then enter data or change data.

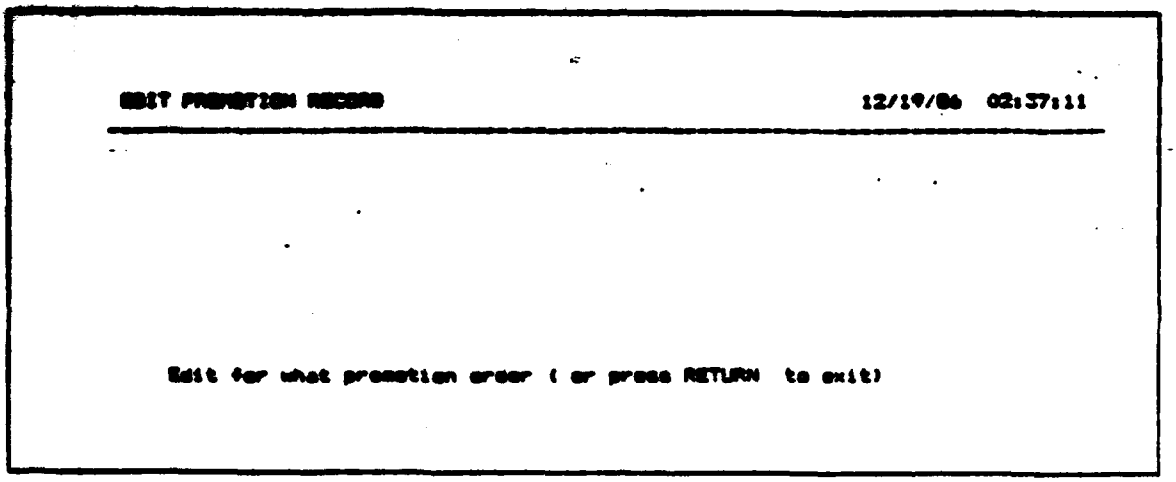


Figure 7.24 Query to change promotion record.

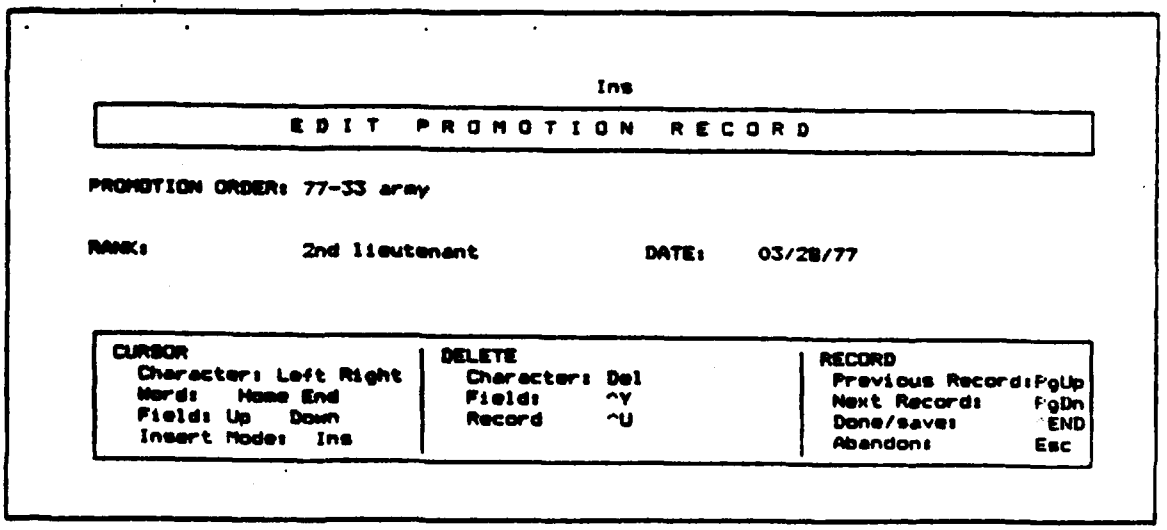


Figure 7.25 Screen for changing for promotion record.

If you select option "B" from the add and edit promotion record sub-menu, you will see the following screen(Figure 7.26). This will allow you to add and edit any fields in the PROMOTE.DBF. You must know the service number and promotion order (in Appendix) exactly. After entering the service number and promotion order, the following screen(Figure 7.27) will appear, and then you may enter or change data.

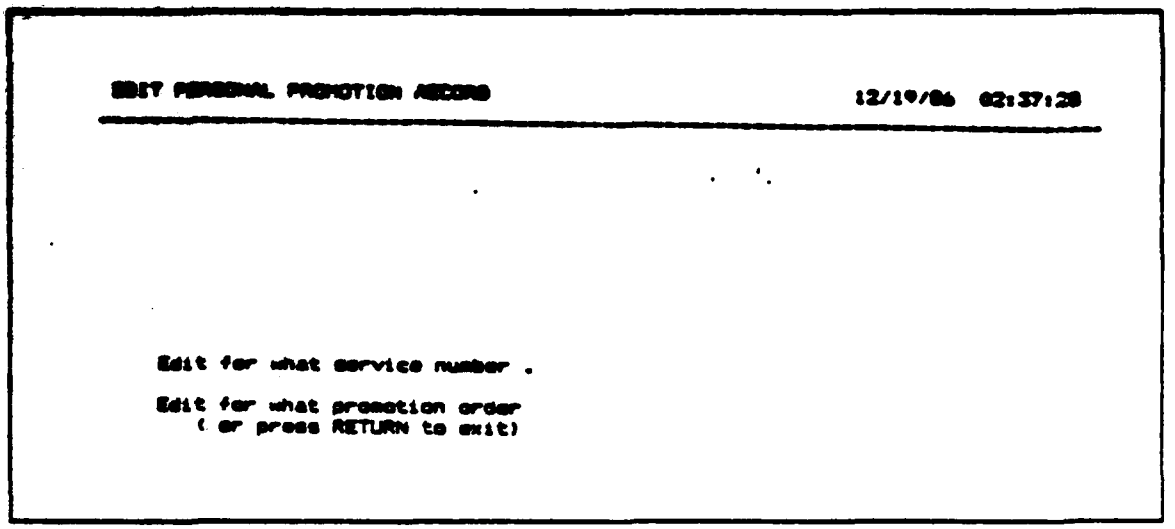


Figure 7.26 Queries to change personal promotion record.

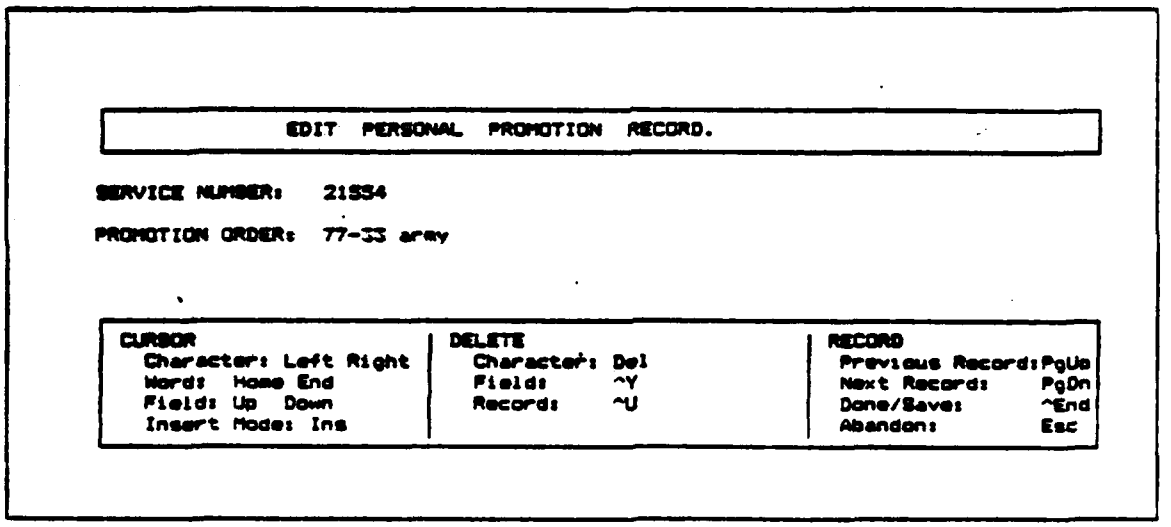


Figure 7.27 Screen for changing personal promotion record.

5. Select option "E" to change award and punishment records

This will allow you to change award and punishment records. The following sub-menu screen(Figure 7.28) will appear on the screen.

If you select option "A" from the award and punishment sub-menu, you will see the following screen(Figure 7.29). This will allow you to add and change any fields in the A_P_P.DBF you must know the award and punishment name (in Appendix)

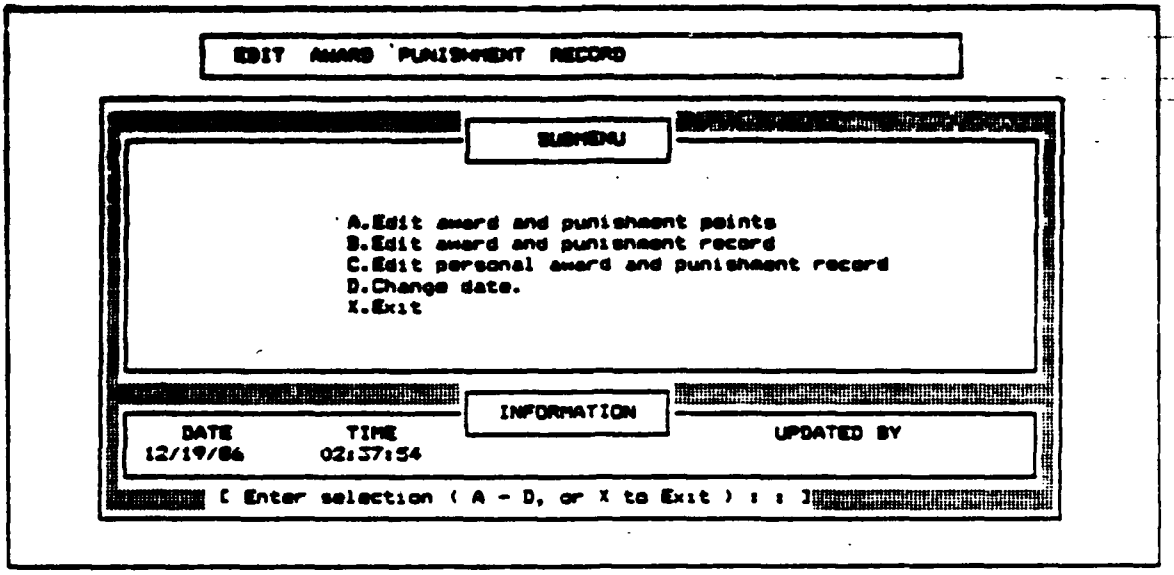


Figure 7.28 Submenu for changing award and punishment records..

exactly. After entering the award and punishment name, the following screen (Figure 7.30) will appear, and then you may enter or change data.

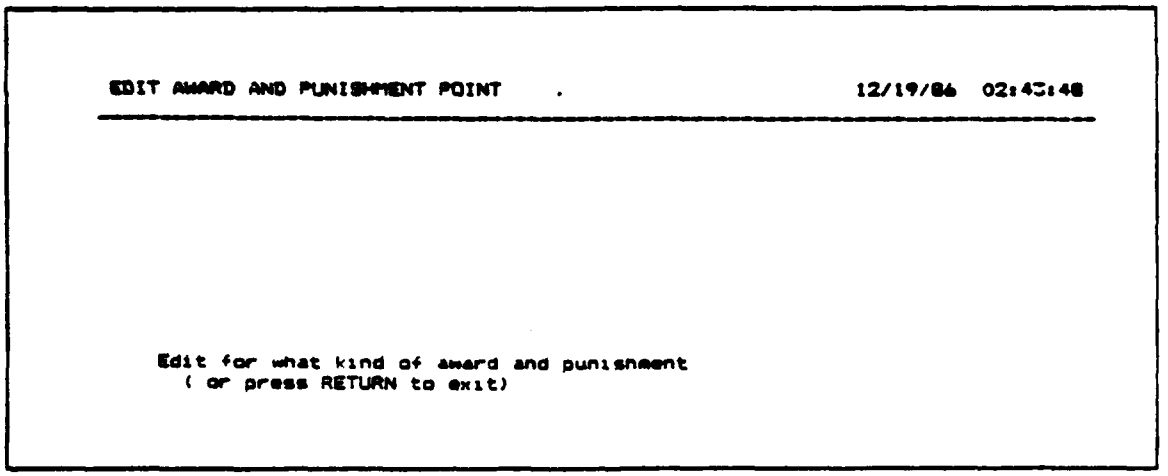


Figure 7.29 Query to change award and punishment points.

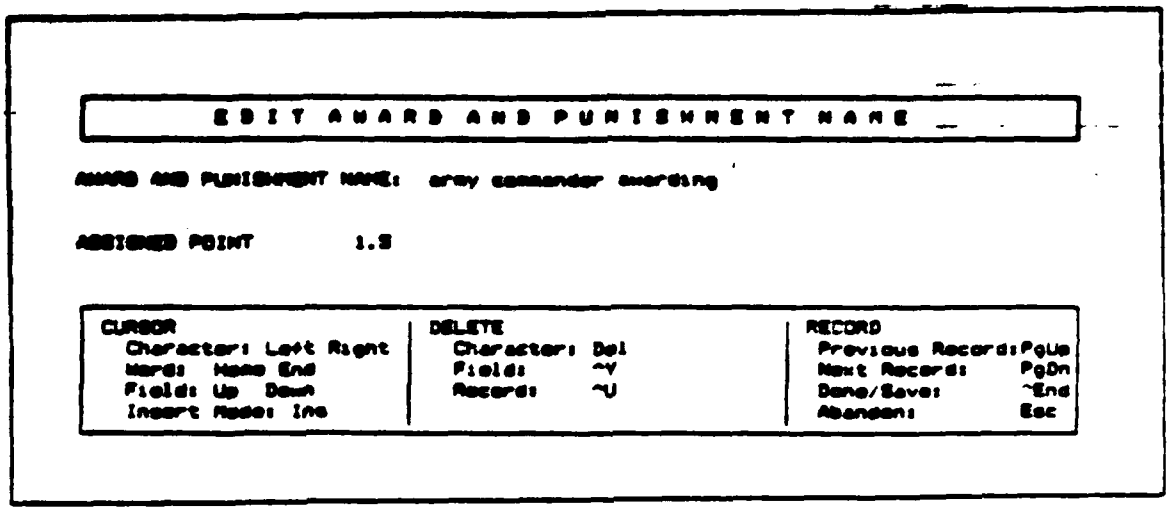


Figure 7.30 Screen for changing award and punishment points.

If you select option "B" from the award and punishment sub-menu, you will see the following screen(Figure 7.31). This will allow you to add and edit any fields in AWARDPUN.DBF file. You must know the personnel order (in Appendix) exactly. After entering the personnel order, the following screen(Figure 7.32) will appear, and then you may enter or change data.

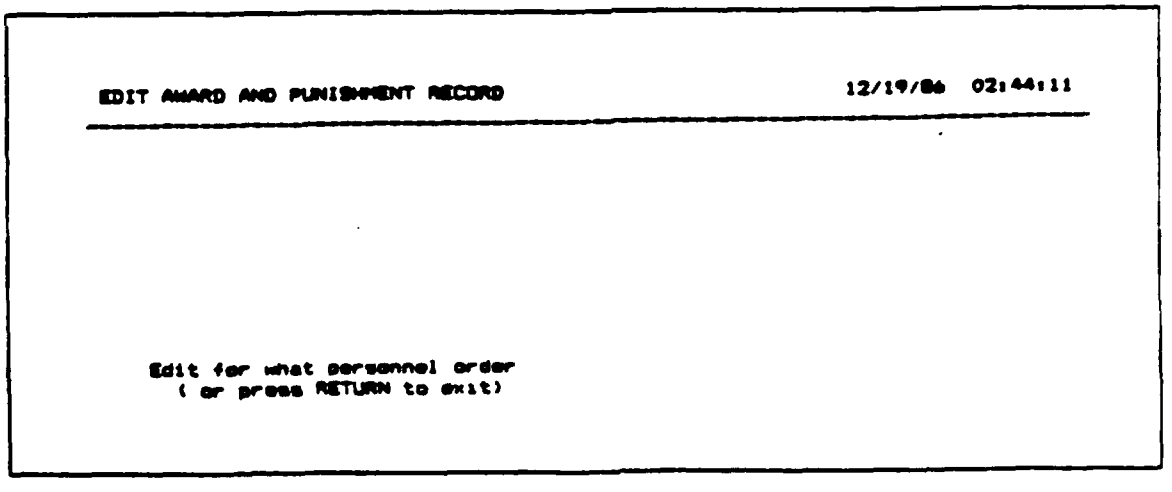


Figure 7.31 Query to change award and punishment record..

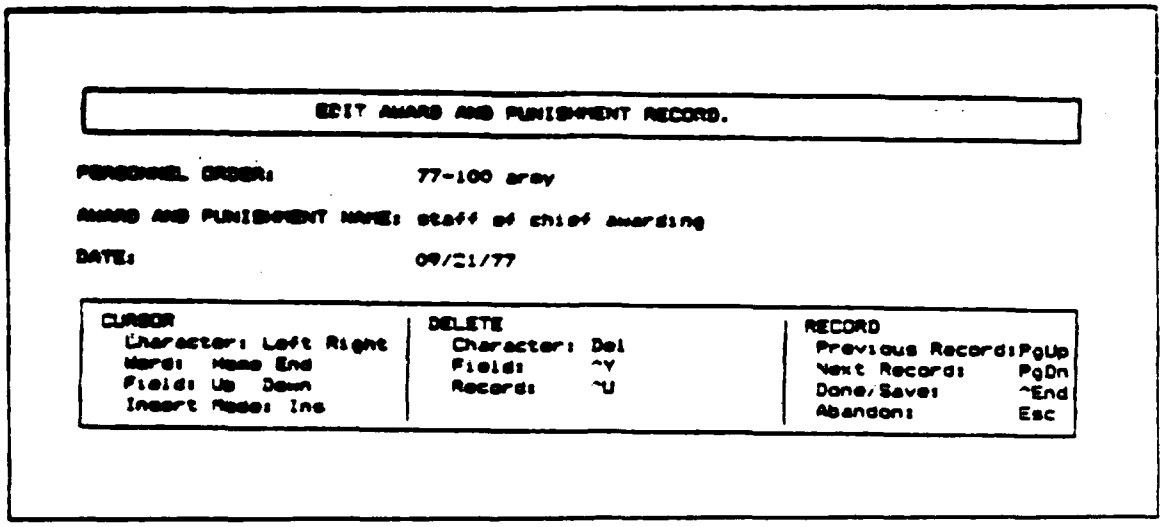


Figure 7.32 Screen for changing award and punishment record.

If you select option "A" from the award and punishment sub-menu, you will see the following screen(Figure 7.33). This will allow you to add and edit any fields in A_P_MN.DBF file. You must know the service number and personnel order(in Appendix) exactly. After entering the service number and personnel order, the following screen(fig 7.34) will appear, and then you may enter or change data

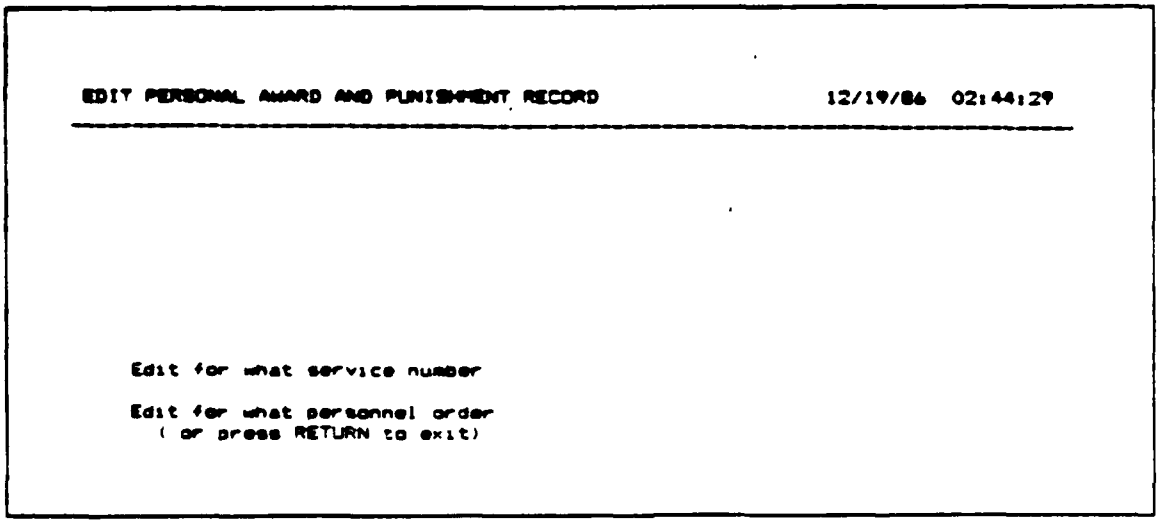


Figure 7.33 Query to change personal award and punishment.

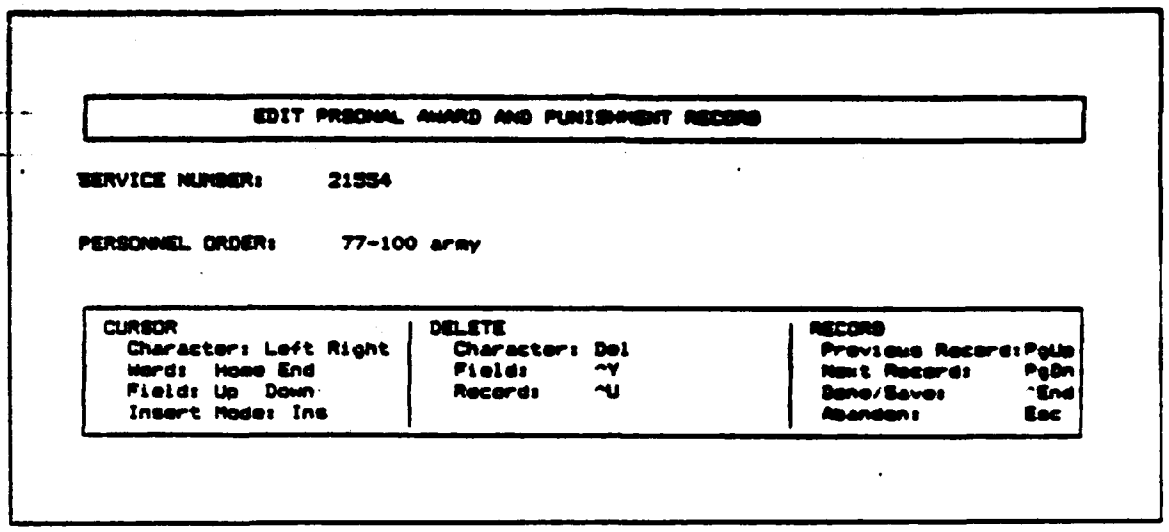


Figure 7.34 Screen for changing personal award and punishment record.

6. **Select option "F" to change performance evaluation record**

This will allow you to add and change any fields in the P_EVAL.DBF file. The following screen(Figure 7.35) will appear.

You must know the service number and evaluation date, failure to do so may result in the wrong information. After entering the service number and evaluation date, the following screen(Figure 7.36) will appear, and then you may enter or change data.

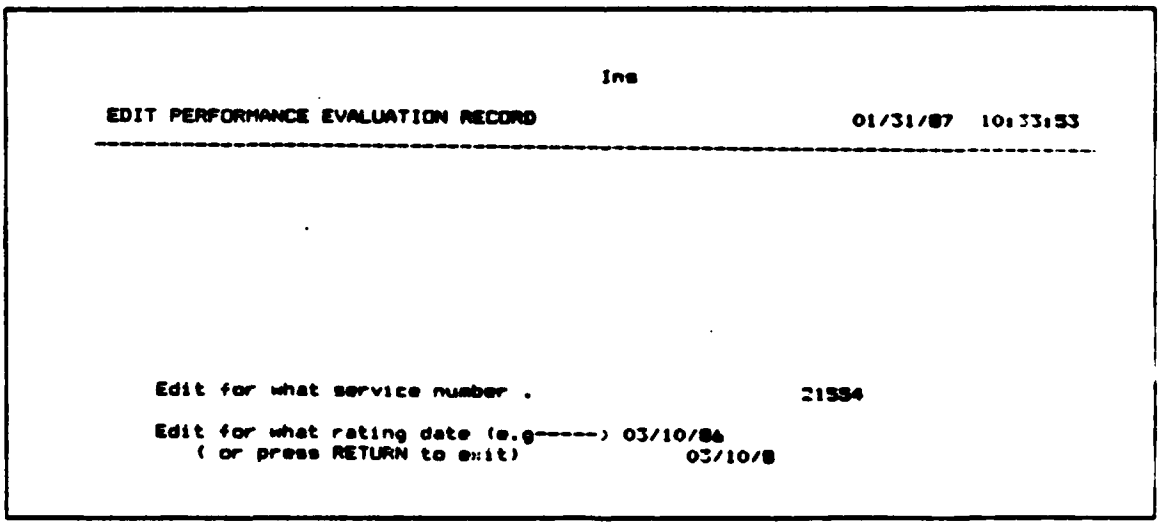


Figure 7.35 Queries to change performance evaluation records.

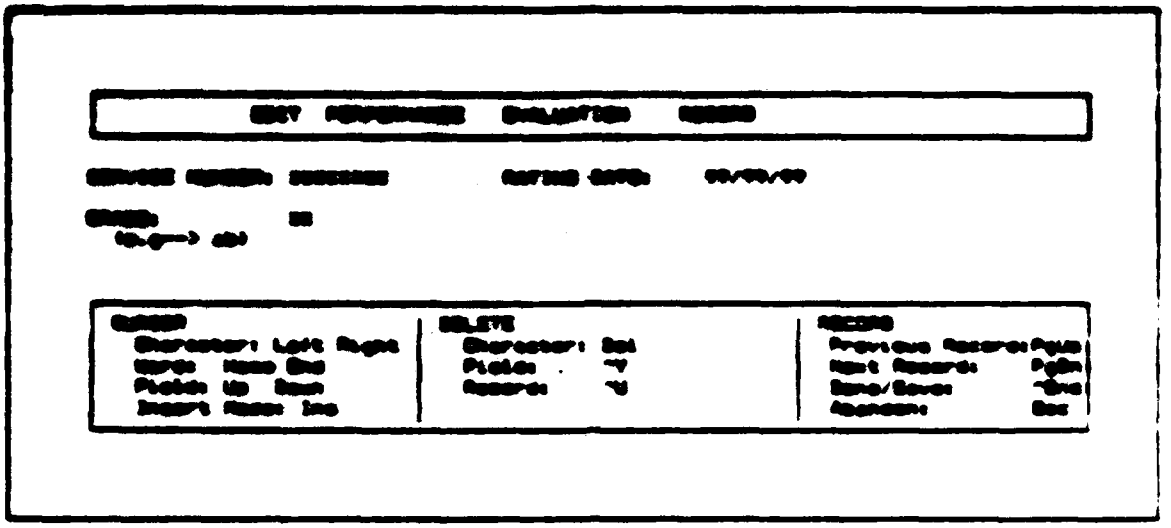


Figure 7.36 Screen for changing performance evaluation records.

7. Select option"C" to change the assignment record

This will allow you to add and change any fields in the CAREER.DBF file. The following screen(Figure 7.37) will appear on the screen. you must know the service number and personnel order (in Appendix) exactly. After entering the service number and personnel order, the following screen(Figure 7.38) will appear, and then you may enter or change data.

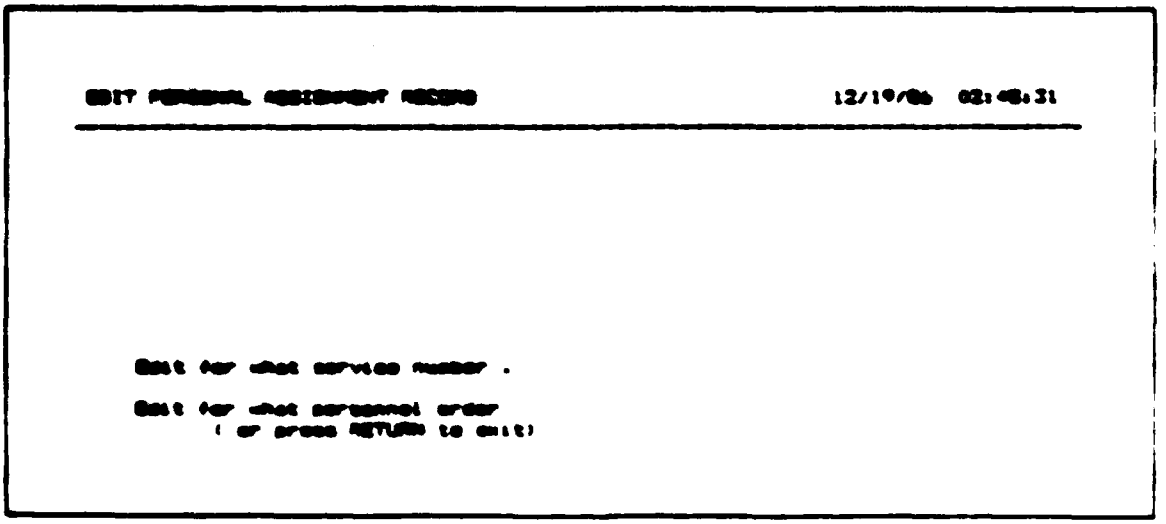


Figure 7.37 Queries to change personal assignment record.

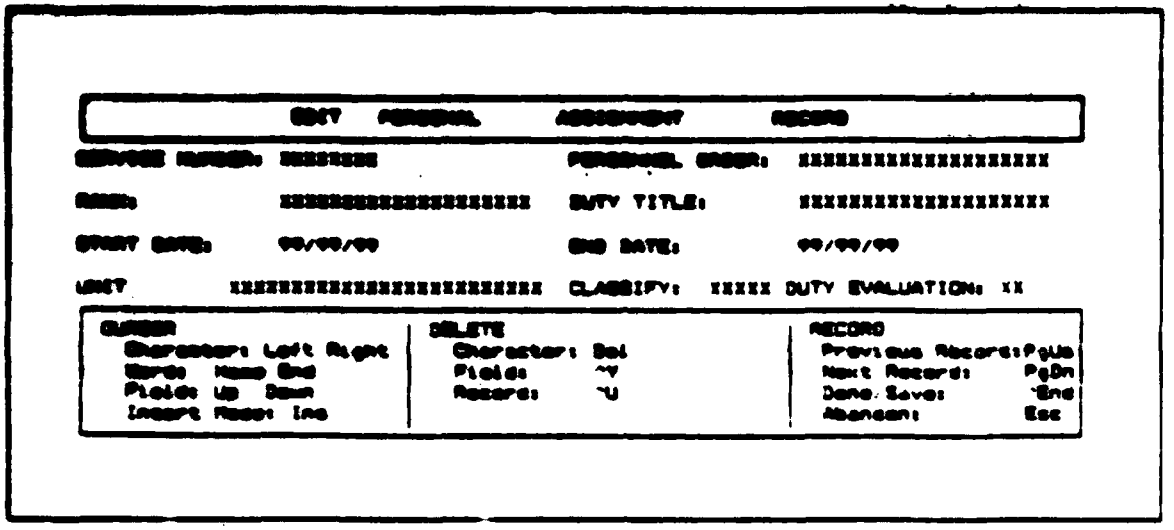


Figure 7.38 Screen for changing personal assignment record.

F. DATA DICTIONARY

The main menu of the PMS system (OPTION F) allows you to access the data dictionary. Through the dictionary menu, you can find out what a variable name is and how to enter it into the computer (option A), find out the structure of commonly used data files (option B), find out where a variable is used in these files and modules (option C) and find out other features of the dictionary with an exit back to main PMS menu (option D). By doing this, you have more interactive flexibility when using the system. You must still have some idea about what a variable is named before you can list the file structure and retrieve the exact variable name. The following screen (Figure 7.39) will appear.

1. Select option "A" to find out element name in the file

This allows you to ask questions about element and how they are entered into the computer. For example, you know the variable name or at least a few letters of it. You will see DO YOU WANT A PRINTOUT? Y OR N If you desire a hardcopy printout of the information, you would ready your printer at this time and enter a capital Y. If you do not desire a printout, enter N. The information you will receive will be element name, full name, type, frequency of update and comments. If you entered a few letters of the name, you may get several element names and values. All of the element names will appear first, then all of the full names will appear second, and so forth. You can then choose the one you want and get a clean copy of the exact

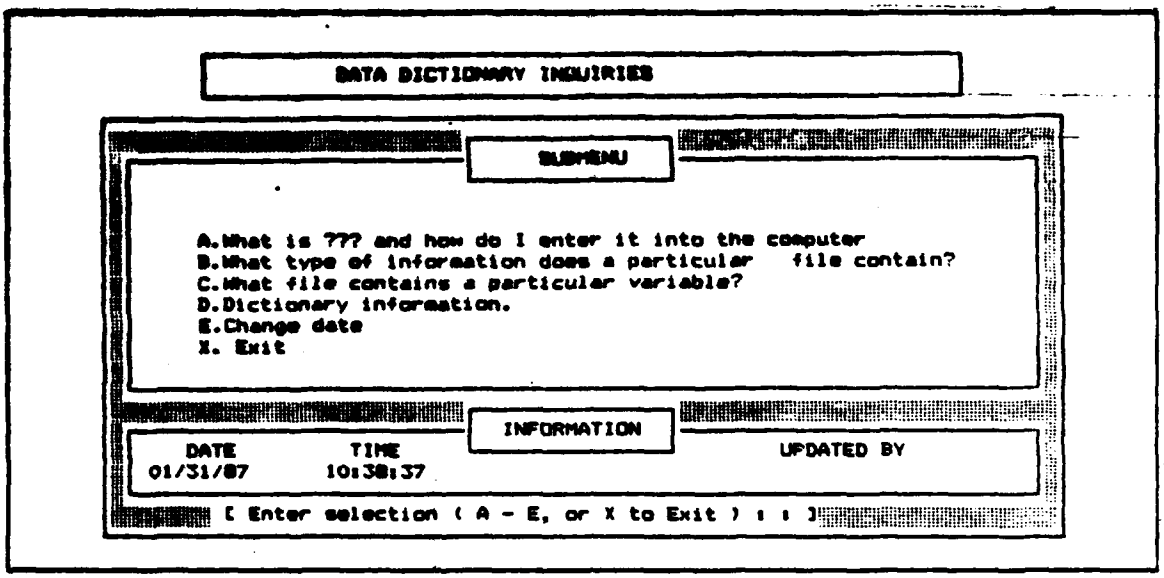


Figure 7.39 Submenu of data dictionary.

element name, if desired. REQUIRED INFORMATION-- The element name or at least a few letters of it. You may obtain this first through dictionary option 2 if you know what file it is used in.

2. **Select option "B" to find out used data files**

This allows you to discover how certain files are structured. The options are alphabetic and they represent the commonly used data files in PMS. You will see the following screen (Figure 7.40).

You will then enter the letter of your selection. If you enter something besides the menu options, you will see the same screen. If you want to leave this portion of the dictionary, select the letter for "X". You will then be asked if you want a printout. If you do, you ready your printer at this time, and then enter a capital Y. If you do not want a printout, type N.

This option allows you to discover the exact variable names (listed as fields) used in a file.

3. **Select option "C" to find out used modules**

This portion of the dictionary allows you to make queries about where certain variables are used throughout PMS. For example, if you wanted to know where CNAME was used, you would enter the variable name (or what you remembered of it) when you see PLEASE ENTER THE VARIABLE NAME-- you will then be asked if

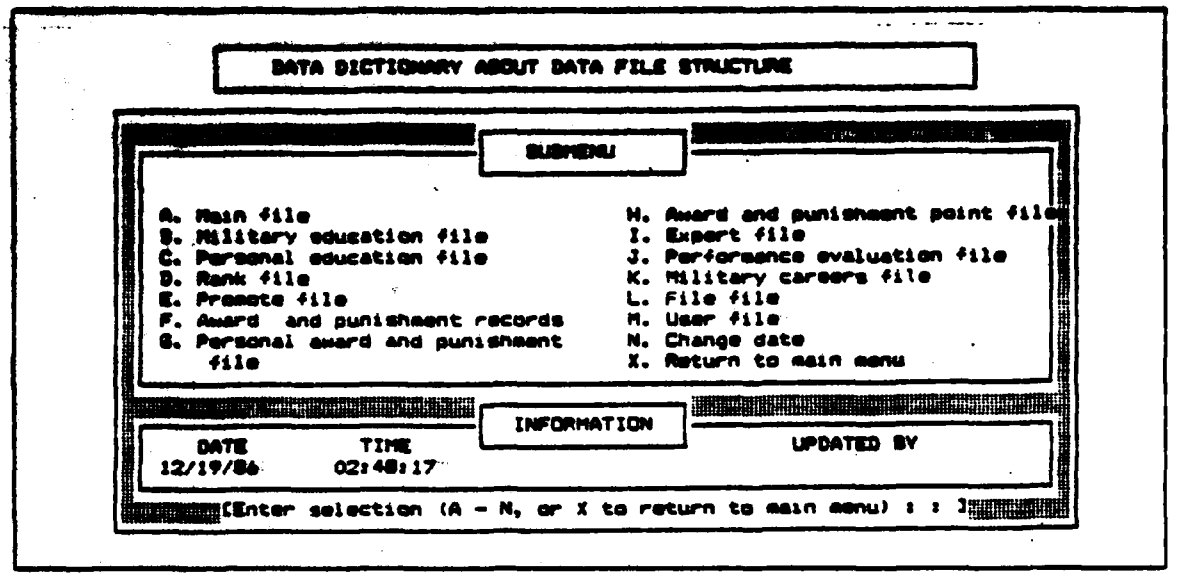


Figure 7.40 Submenu to detect data structure.

you want a printout. If you do, ready your printer first, then type a capital Y. If not, type N.

The information you receive will list the name of the file or module where it is used, then the type (file or module). BE CAREFUL-- If you listed only a portion of the variable name, you may get a list of where all the variable names are used. However, by judiciously using this option with OPTIONS A and B, you can have the flexibility of discovering where variables are used (OPTION C), the exact variable(field) name(OPTION B),and how to enter it into the computer (OPTION A). REQUIRED INFORMATION--

The variable name or at least a few letters of it. You may obtain this first through dictionary option B if you know at least one file in which it is used.

Each portion of the data dictionary has a menu to assist you in entering the required data for your inquiry. However, if you find you can't get out of the dictionary for some reason, type HELP in all capital letters when asked for a variable name. Then answer N to the question about wanting a printout. This will allow you to return to the main dictionary menu and then return back to the main PMS menu through OPTION D or OPTION X

4. Select option "D" to find out the other features

This allows you to exit the dictionary menu and return to the main PMS menu. Before leaving the dictionary, it gives you some further information on the files that exist in the dictionary.

Some of the data dictionary files are not accessible by you, and can only be accessed by someone having knowledge of DBASE III+. You can also only read data out of the dictionary: database personnel must add new entries to the dictionary through DBASE III+. The primary reason for this is the fact that most of these files contain information generally only used by database personnel.

Each of the files is shown below with a brief description of what it contains:

- CONTAINS-- Files which contain elements(variables)
- FILE-- Describes the files in PMS.
- ELEMENT-- The variable names and information about them.
- PROCESSES-- The programs and modules which process the elements and files.
- PROGRAMS-- The programs which control the operations of each of the menu options in the PMS system.
- AUTOFILE-- Describes the auto files in PMS.
- USER-- Describes the users of PMS.

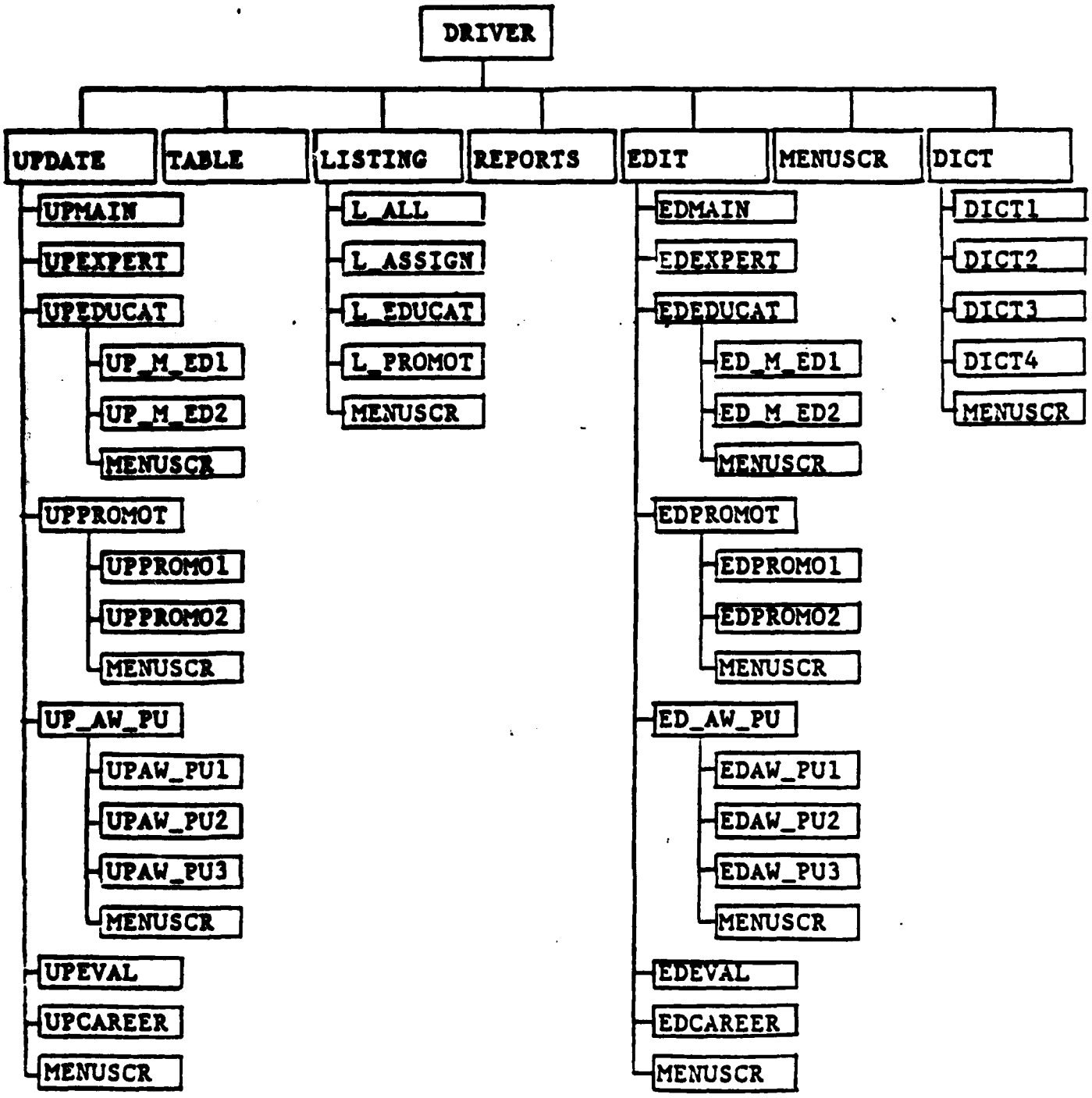
VIII. CONCLUSION

This thesis has focused on application of a data base system for a Republic of Korea military personnel management system. In order to reduce expenditures and manpower for personnel management and to increase the ability of combat soldiers, it is very important for the Korean military to apply the computer system for personnel management. Since we use dBASE III+ for application program in this thesis, we should use relational data base model. Therefore, we reviewed the basic knowledge about a data base system in Chapter II and discussed the general concept of relational model in Chapter III. After that, we focused precisely about the design problem of the relational model in Chapter IV. In Chapter V, we discussed the practical system analysis and relational database design for a Korean Army personnel management system. In order to maintain and use the data base system effectively, we discussed the relational implementation in Chapter VI. Especially, in Chapter VII, processing procedure to access actual program in Appendix A was shown. When we construct a data file to access this program we should consider theoretical problems that are discussed in Chapter IV. In prototyping personnel data base in this thesis, all data items such as career, assignment, education, etc. is based on the Korean army personnel record. Because of the military security problem we use artificial sample data for prototyping.

In order to strengthen the readiness of the Korean military under the alert states, it is imperative that personnel management be performed very efficiently. A most important consideration in data base development is to store data so that it can be used for a wide variety of applications and can be changed quickly and easily. In order to perform these functions, the data should be independent and functionally dependent on key values. It should also be possible to query the data base to satisfy user's requirements using application programmer the Database Management System(DBMS) itself. These data items should contain useful information for decision makers to analyze, plan and manage a personnel organization. However, a data base is the interface between people and machines. Data base design is a two-phased process. This thesis examined both logical and physical data base design process, and this process is an iterational process to get closer to an acceptable and optimal design.

As it is discussed in Chapter IV, normal forms can be applied to decrease inefficiency of the relational data base model in the system design process. Finally, we hope this research and sample application can be helpful for Korean military personnel management system.

APPENDIX A
PROGRAM STRUCTURE



APPENDIX B
PROGRAM

1. DRIVER.PRG

```
*************************************************************************
* Module name   :DRIVER.PRG
* Author        :KIM,  SAM NAM
* Date          :31 OCT 86
* Purpose       :This is the DRIVER module for the entire system.
*                It queries the user for task selection and calls
*                the correct module to service that request.
*                When control is returned from the calling module
*                the user is asked if more information is required.
*                If yes,  then the process is repeated,  else,  the
*                program terminates.
* Called by     : DRIVER is called at system startup
* Modules called :MENUSCR,TABLE,LISTING,REPORTS,UPDATE,EDIT,DICT
* Variables used :
*         Global : i : holds the value of the user input.
*                  today: holds date
*         Local  : none
*************************************************************************
*          Set up loop for presenting menu.
*************************************************************************
*Close all open files
CLEAR ALL
*---------- Set working working environment
SET TALK OFF
SET ESCAPE OFF
SET BELL OFF
SET HEADING OFF
SET HELP OFF
SET MENU OFF
SET SAFETY OFF
SET STATUS OFF
*---------- Create underline variable, Uline.
uline=REPLICATE ("_", 80)
*---------- Create memory variable for today's date.
today=DATE()
*************************************************************************
* This sets up the CRASH.TXT file which records all actions so
* that if the system crashes,  the data base can be recreated. This
* file is deleted if the system terminates normally.
*************************************************************************
SET ALTE TO CRASH
SET ALTE ON
DO WHILE .T.
   * DO WHILE .T. means DO WHILE TRUE i.e. DO FOREVER
   * The DO WHILE will be terminated by an EXIT command
   * Clear the screan and display the main menu
   CLEAR
   DO MENUSCR
   @ 2,11 SAY "  O F F I C E R   P E R S O N N E L   M A N A G E M "
   @ 2,62 SAY "E N T"
   @ 6,36 SAY "MENU"
   @ 19,33 SAY "INFORMATION"
   @ 8,12 SAY "A. PERSONNEL ALLOTMENT TABLE."
   @ 9,12 SAY "B. LISTING."
   @ 10,15 SAY " Any query that requires a listing or information"
   @ 11,12 SAY "C. REPORT(personal record card)."
   @ 12,12 SAY "D. UPDATE."
   @ 13,12 SAY "E. EDIT."
   @ 14,12 SAY "F. DICTIONARY."
```

```
   @ 15,12 SAY "G. CHANGE DATE."
   @ 16,12 SAY "X. EXIT."
   @ 20,8 SAY "DATE          TIME"
   @ 20,55 SAY "UPDATED BY"
   @ 21,5 SAY today
   @ 21,19 SAY TIME()
*  @ 21,55 SAY gname
   @ 23,10 SAY " [ Enter selection ( A - G, or X to Exit ) : : ]"
   DO WHILE .T.
      i=0
      DO WHILE i=0
         i=INKEY()
         @ 21,19 SAY TIME()
         @ 23,54 SAY ""
         IF UPPER(CHR(i))$"ABCDEFGX"
               EXIT
         ENDIF
         i=0
      ENDDO
      @ 23,54 SAY UPPER(CHR(i))
      IF .NOT. CHR(i)$"Gg"
         EXIT
      ENDIF
      SET COLOR TO N/W
      @ 19,33 SAY "INFORMATION"
      @ 15,12 SAY "G. CHANGE DATE."
      SET COLOR TO W/N
      @ 21,5  GET today
      READ
      @ 21,5 SAY today
      @ 19,33 SAY "INFORMATION"
      @ 15,12 SAY "G. CHANGE DATE."
      @ 23,54 SAY " "
   ENDDO
   DO CASE
      CASE CHR(i)$ "Xx"
         RELEASE i,today
         SET TALK ON
         SET ESCAPE ON
         SET BELL ON
         SET HEADING ON
         SET HELP ON
         SET MENU ON
         SET SAFETY ON
         SET STATUS ON
         CLEAR
         RETURN
      CASE CHR(i)$"Aa"
         DO TABLE
      CASE CHR(i)$"Bb"
         DO LISTING
      CASE CHR(i)$"Cc"
         DO REPORTS
      CASE CHR(i)$"Dd"
         DO UPDATE
      CASE CHR(i)$"Ee"
         DO EDIT
      CASE CHR(i)$"Ff"
         DO DICT
   ENDCASE
ENDDO
SET ALTE OFF
CLEAR ALL
CLOSE ALTE
ERASE CRASH.TXT
*---------- when done,return to main menu
RETURN
```

a. MENUSCR.PRG

```
*************************************************************************
* Module name          : MENUSCR.PRG
* Author               : KIM,  SAM NAM
* Date                 : 30 OCT 86
* Purpose              : Menu screen
* Modules called       : None
* Variables used       : Local  : none
*                        Grobal : today : holds date
*************************************************************************
*           Set up presenting menu.
*************************************************************************
@ 1,9 TO 3,69
@ 4,1 TO 24,77 DOUBLE
@ 6,3 TO 17,75
@ 5,30 TO 7,46 DOUBLE
@ 6,31 SAY SPACE(15)
@ 19,3 TO 22,75
@ 18,30 TO 20,46 DOUBLE
@ 19,31 SAY SPACE(15)
@ 5,2 SAY REPLICATE(CHR(176),28)
@ 6,2 SAY CHR(176)
@ 7,2 SAY CHR(176)
@ 8,2 SAY CHR(176)
@ 9,2 SAY CHR(176)
@ 10,2 SAY CHR(176)
@ 11,2 SAY CHR(176)
@ 12,2 SAY CHR(176)
@ 13,2 SAY CHR(176)
@ 14,2 SAY CHR(176)
@ 15,2 SAY CHR(176)
@ 16,2 SAY CHR(176)
@ 17,2 SAY CHR(176)
@ 18,2 SAY REPLICATE(CHR(176),28)
@ 19,2 SAY CHR(176)
@ 20,2 SAY CHR(176)
@ 21,2 SAY CHR(176)
@ 22,2 SAY CHR(176)
@ 23,2 SAY REPLICATE(CHR(176),75)
@ 22,76 SAY CHR(176)
@ 21,76 SAY CHR(176)
@ 20,76 SAY CHR(176)
@ 19,76 SAY CHR(176)
@ 18,47 SAY REPLICATE(CHR(176),30)
@ 17,76 SAY CHR(176)
@ 16,76 SAY CHR(176)
@ 15,76 SAY CHR(176)
@ 14,76 SAY CHR(176)
@ 13,76 SAY CHR(176)
@ 12,76 SAY CHR(176)
@ 11,76 SAY CHR(176)
@ 10,76 SAY CHR(176)
@ 9,76 SAY CHR(176)
@ 8,76 SAY CHR(176)
@ 7,76 SAY CHR(176)
@ 6,76 SAY CHR(176)
@ 5,47 SAY REPLICATE(CHR(176),30)
@ 19,33 SAY "INFORMATION"
@ 20,8 SAY "DATE          TIME"
@ 20,55 SAY "UPDATED BY"
@ 21,5 SAY today
@ 21,19 SAY TIME()
*@ 21,52 SAY gname
RETURN
```

2. TABLE.PRG

```
*************************************************************************
* Module name   :TABLE.PRG
* Author        :KIM,  SAM NAM
* Date          :1 DEC 86
* Purpose       :This is the DRIVER module for personnel allotment.
*                When control is returned from the called module
*                the user is asked if more information is required
*                and the process is repeated,  or control is passed
*                back to the DRIVER module.
* Called by     : DRIVER
* Modules called :MENUSCR
* Variables used :
*          Global : i : holds the value of the user input.
*                   today: holds date
*          Local  : none
*************************************************************************
*          Set up allotment table.
*************************************************************************
CLEAR
DO WHILE .T.
   CLEAR
   DO MENUSCR
   @ 2,15 SAY "P E R S O N N E L   A L L O T M E N T "
   @ 6,35 SAY "TABLE"
   @ 19,33 SAY "INFORMATION"
   @ 10,18 SAY "COMPANY GRADE OFFICER:"
   @ 11,18 SAY "FIELD GRADE OFFICER :"
   @ 12,18 SAY "GENERAL GRADE OFFICER:"
   @ 14,18 SAY "  TOTAL                :"
   @ 16,18 SAY "**** X. Return to main menu *****"
   @ 20,8 SAY "DATE          TIME"
   @ 20,55 SAY "UPDATED BY"
   @ 21,5 SAY today
   @ 21,19 SAY TIME()
  * @ 21,55 SAY gname
   @ 23,10 SAY "[       Enter  X  to  Exit                  : : ]"
   USE member INDEX member
        FIND warrent officer
        COUNT WHILE ranks = "warrent officer" TO xx1
        FIND 2nd lieutenent
        COUNT WHILE ranks = "2nd lieutenent" TO xx2
        FIND 1st lieutenent
        COUNT WHILE ranks = "1st lieutenent" TO xx3
        FIND captain
        COUNT WHILE ranks = "captain" TO xx4
        FIND major
        COUNT WHILE ranks = "major" TO xx5
        FIND lieutenent colonel
        COUNT WHILE ranks = "lieutenent colonel" TO xx6
        FIND colonel
        COUNT WHILE ranks = "colonel" TO xx7
        FIND brigader general
        COUNT WHILE ranks = "brigader general" TO xx8
        FIND major general
        COUNT WHILE ranks = " major general" TO xx9
        FIND lieutenent general
        COUNT WHILE ranks = "lieutenent general" TO xx10
        FIND general
        COUNT WHILE ranks = "general" TO xx11
        company = xx1 + xx2 + xx3 + xx4
        field = xx5 + Xx6 + xx7
        gener = xx8 + xx9 + xx10 + xx11
        addsum = company + field + gener
        @ 10,45 SAY company
        @ 11,45 SAY field
        @ 12,45 SAY gener
        @ 14,45 SAY addsum
```

```
      DO WHILE .T.
         i=0
         DO WHILE i=0
            i=INKEY()
            @ 21,19 SAY TIME()
            @ 23,54 SAY ""
            IF UPPER(CHR(i))$"X"
                  EXIT
            ENDIF
            i=0
         ENDDO
         @ 23,54 SAY UPPER(CHR(i))
         IF .NOT. CHR(i)$" "
            EXIT
         ENDIF
         READ
      ENDDO
      DO CASE
         CASE CHR(i)$ "Xx"
            CLEAR
            CLOSE DATABASES
            RETURN
      ENDCASE
ENDDO
*---------- when done,return to main menu
```

3. LISTING.PRG

```
*************************************************************************
* Module name   :LISTING.PRG
* Author        :PARK,  JAE BOCK
* Date          :1 nov 86
* Purpose       :This is the DRIVER module for the listing system.
*                It queries the user for task selection and calls the
*                required modules.
*                When control is returned from the called module,
*                the user is asked if more information is required
*                and the process is repeated,  or control is passed
*                back to the DRIVER module.
* Called by     : DRIVER
* Modules called :MENUSCR,  L_ALL,  L_ASSIGN,  L_EDUCAT,  L_PROMOT
* Variables used :
*          Global : i : holds the value of the user input.
*                   today: holds date
*          Local  : none
*************************************************************************
*           Set up loop for presenting menu.
*************************************************************************
CLEAR
DO WHILE .T.
   CLEAR
   DO MENUSCR
   @ 2,32 SAY "L I S T I N G "
   @ 6,34 SAY "SUBMENU"
   @ 19,33 SAY "INFORMATION"
   @ 8,12 SAY "A. List all officers."
   @ 9,12 SAY "B. List all new officers for assignment."
   @ 10,12 SAY "C. List all officers educated. "
   @ 11,12 SAY "D. List all officers for promotion test."
   @ 12,12 SAY "E. Change date"
   @ 14,12 SAY "X. Return to main menu"
   @ 20,8 SAY "DATE          TIME"
   @ 20,55 SAY "UPDATED BY"
   @ 21,5 SAY today
   @ 21,19 SAY TIME()
 * @ 21,55 SAY gname
   @ 23,10 SAY " [ Enter selection ( A - E, or X to Exit ) : : ]"
   DO WHILE .T.
```

```
        i=0
        DO WHILE i=0
            i=INKEY()
            @ 21,19 SAY TIME()
            @ 23,54 SAY ""
            IF UPPER(CHR(i))$"ABCDEX"
                EXIT
            ENDIF
            i=0
        ENDDO
        @ 23,54 SAY UPPER(CHR(i))
        IF .NOT. CHR(i)$"Ee"
            EXIT
        ENDIF
        SET COLOR TO N/W
        @ 19,33 SAY "INFORMATION"
        @ 12,12 SAY "E. CHANGE DATE"
        SET COLOR TO W/N
        @ 21,5  GET today
        READ
        @ 21,5 SAY today
        @ 19,33 SAY "INFORMATION"
        @ 12,12 SAY "E. Change date"
        @ 23,54 SAY " "
    ENDDO
    DO CASE
        CASE CHR(i)$ "Xx"
           CLEAR
           RETURN
        CASE CHR(i)$"Aa"
           DO L_ALL
        CASE CHR(i)$"Bb"
           DO L_ASSIGN
        CASE CHR(i)$"Cc"
           DO L_EDUCAT
        CASE CHR(i)$"Dd"
           DO L_PROMOT
     ENDCASE
ENDDO
*------------ when done,return to main menu
RETURN
```

a. L_ALL.PRG

```
*************************************************************************
* Module name   :L_ALL.PRG
* Author        :KIM,   SAM NAM
* Date          :15 DEC 86
* Purpose       :This module provides the listing service required
*                to list all members.
* Called by     : LISTING
* Modules called : none
* Variables used :
*          Global : none
*          Local  : park : holds  values of user answer.
*************************************************************************
*          Set up loop for presenting menu.
*************************************************************************
CLEAR
SET TALK OFF
park = "x"
park = SPACE(1)
@ 15,5 SAY "Send data to printer ? (Y/N)" GET park
READ
SELECT 1
USE main INDEX MAIN
SELECT 2
USE member
JOIN WITH main TO all FOR sns = A->sn ;
FIELDS ranks,sns,A->name,A->c_type,org_branch,p_order
```

```
USE all
IF [illegible] .NT.
   [illegible] exact on
   [illegible] ranks,ans,name,c_type
   [illegible] ORDER ON
ELSE
   LIST ranks,ans,name,c_type
ENDIF
USE
CLOSE DATABASES
*---------- When done,return to main menu.
RETURN
    b. L_ASSIGN.PRG
*************************************************************************
* Module name  :L_ASSIGN.PRG
* Author       :PVT[illegible]  JAE BOCK
* Date         :1 DEC 86
* Purpose      :This is the LISTING module for the listing system.
*               This module is used who the user requires a new
*               officers list for assignment.
* Called by    : LISTING
* Modules called :
* Variables used :
*         Global : today: holds date
*         Local  : rank : holds the values of the user input,
*                         accept the military rank.
*                  order: holds the values of the user input,
*                         accepts the personel order.
*************************************************************************
*         Set up loop for presenting menu.
*************************************************************************
CLEAR
USE all INDEX p_all
rank = "x"
order = "x"
DO WHILE order # " " .OR. rank # " "
   *------------ Find out the personnel order and rank.
   CLEAR
   @ 2,1 SAY "LIST ALL NEW OFFICERS' RECORD"
   @ 2,40 SAY today
   @ 2,70 SAY PVT[illegible]()
   @ 3,0 SAY ULINE
   ?
   ?
   @ 5,0 CLEAR
   *----------- Get proposed personnel order and rank.
   order = SPACE(20)
   rank = SPACE(20)
   @ 15,5 SAY "List for what personnel order( or press RETURN
   to exit)"
   @ 16,5 SAY "( e.g ----->  86-35 ARMY )" GET order
   READ
   @ 17,5 SAY "List for what rank   (or press RETURN to exit)"
   @ 18,5 SAY "(e.g ----> 1st lieutenent)" GET rank
   READ
   order = order + rank
   order = lower(order)
   SEEK order
   DO CASE
      CASE order = " "
      CLEAR
      CASE FOUND()
        @ 5,0 CLEAR
        STORE " " TO YN,printer
        @ 15,15 SAY "Send to printer ?(Y/N).     ' GET YN PICT '!'
        READ
        *------ Set up printer macro.
        IF YN = "Y"
```

```
            printer = "TO PRINT"
            [illegible]
            [illegible] FORM r_assign FOR p_order + ranks = order &printer
            [illegible]
        CASE .NOT. FOUND()
            [illegible]
            @ 17,5 SAY " There is no " + order
            @ 24,5 SAY "PRESS ANY KEY TO TRY AGAIN....."
            WAIT [illegible](7)
            [illegible]
        [illegible]
[illegible]
*------ When done, return to listing menu.
RETURN
```

e. L_EDUCAT.PRG

```
*************************************************************************
* Module name   :L_EDUCAT.PRG
* Author        :PEEK,  JAE HOCK
* Date          :1 DEC 86
* Purpose       :This is the LISTING module for the listing system.
*                This module is used when the user requires the list
*                of officers educated in a particular courses.
* Called by     : LISTING
* Modules called :none
* Variables used :
*         Local  : names : holds the value of the user input.
*                  acceptable values : military education course name.
*        Global : today: holds date
*************************************************************************
*          Set up loop for presenting menu.
*************************************************************************
CLEAR
USE j_educat INDEX j_educat
[illegible]
DO WHILE names # " "
    *------------ Find out what course name to list.
    CLEAR
    @ 1,60 SAY today
    @ 1,70 SAY TIME()
    @ 3,0 SAY WLINE
    ?
    ?
    *------------ Get proposed course name.
    names = SPACE(20)
    @ 15,5 SAY "List for what course name( or press RETURN;
to exit)"
    @ 16,5 SAY "( e.g -->INFANTRY O.A.C0234 )" GET names
    READ
    names = lower(names)
    SEEK names
    DO CASE
        CASE names = " "
        CLEAR
        CASE FOUND()
            @ 5,0 CLEAR
            STORE " " TO YN,printer
            @ 15,15 SAY "Send to printer ?(Y/N).     " GET YN PICT "!"
            READ
            *------- Set up printer macro.
            IF YN = "Y"
                Printer = "TO PRINT"
            ENDIF
            REPORT FORM r_educat FOR cname = names &printer
            WAIT
        CASE .NOT. FOUND()
            CLEAR
            @ 17,5 SAY " There is no " + names
            @ 24,5 SAY "PRESS ANY KEY TO TRY AGAIN....."
```

```
        ? CHR(7)
        WAIT
      ENDCASE
ENDDO
*------ When done, return to listing menu.
RETURN
```

d. L_PROMOT.PRG

```
*************************************************************************
* Module name   :L_PROMOT.PRG
* Author        :PARK,  JAE BOCK
* Date          :1 DEC 86
* Purpose       :This is the LISTING module for the listing system.
*                This module is used when the user requires the
*                officers list for promotion test.
* Called by     : LISTING
* Modules called :
* Variables used :
*         Local  : rank: holds the value of the user input,
*                        acceptable values: military rank.
*                  year: holds the value of the user input,
*                        acceptable values: promotion year.
*         Global : today : holds the date
*************************************************************************
*          Set up loop for presenting menu.
*************************************************************************
CLEAR
USE all INDEX r_all
rank = "x"
years ="x"
DO WHILE  rank # " " .OR. years # " "
   *------------ Find out  rank and promotion year.
   CLEAR
   @ 2,1 SAY "LIST ALL OFFICERS FOR PROMOTION TEST."
   @ 2,60 SAY today
   @ 2,70 SAY TIME()
   @ 3,0 SAY ULINE
   ?
   ?
   @ 5,0 CLEAR
   *----------- Get proposed  and rank.
   rank = SPACE(20)
   years = SPACE(2)
   @ 15,5 SAY "List for what rank   (or press RETURN to exit)"
   @ 16,5 SAY "(e.g ----> 1st lieutenent)" GET rank
   READ
   @ 17,5 SAY "What is a promotion year?(or press RETURN to exit)"
   @ 18,5 SAY "(e.g ----------------------> 82)" GET years
   READ
   rank = LOWER(rank)
   years = LOWER(years)
   SEEK rank
   DO CASE
      CASE rank = " " .OR. years = " "
      CLEAR
      CASE FOUND()
        @ 5,0 CLEAR
        STORE " " TO YN,printer
        @ 15,15 SAY "Send to printer ?(Y/N).    " GET YN PICT "!"
        READ
        *------ Set up printer macro.
        IF YN = "Y"
           Printer = "TO PRINT"
        ENDIF
        REPORT FORM r_promote FOR;
 ranks = rank .AND. SUBSTR(p_order,1,2) <= years &printer
        WAIT
      CASE .NOT. FOUND()
        CLEAR
```

```
        @ 17,5 SAY " There is no  " + rank
        @ 24,5 SAY "PRESS ANY KEY TO TRY AGAIN....."
        ? CHR(7)
        WAIT " "
      ENDCASE
ENDDO
*------ When done, return to listing menu.
RETURN
```

4. REPORTS.PRG

```
*************************************************************************
* Module name   :REPORTS.PRG
* Author        :PARK,  JAE BOCK
* Date          :1 DEC 86
* Purpose       :This is the DRIVER module for the report system.
*                This module is used when the user requires the
*                personnel record card.
* Called by     : DRIVER
* Modules called : none
* Variables used :
*         Global : today: holds date
*         Local  : none
*************************************************************************
*          Set up loop for presenting menu.
*************************************************************************
CLEAR
USE all INDEX all
msn = "x"
DO WHILE  msn # " "
   *------------ Find out  service number.
   CLEAR
   @ 2,1 SAY "PERSONNEL RECORD CARD."
   @ 2,60 SAY today
   @ 2,70 SAY TIME()
   @ 3,0 SAY ULINE
   ?
   ?
   @ 5,0 CLEAR
   *----------- Get proposed  service number.
   msn = SPACE(8)
   @ 15,5 SAY "List for what service number  (or press RETURN to exit)"
   @ 16,5 SAY "(e.g ----> 21554)          " GET msn
   READ
   msn = LOWER(msn)
   SEEK msn
   DO CASE
       CASE msn = " "
          CLEAR
       CASE FOUND()
          @ 5,0 CLEAR
          ACCEPT "Send to printer ?(Y/N)." TO printer
          *------ Set up printer .
          IF printer = "Y"
             SET PRINT ON
          ENDIF
          CLEAR
          REPORT FORM rreport FOR sns = msn
          WAIT " "
          USE EXPERT
          @ 24,5 SAY "EXPERT TITLE "
          LIST expertitle FOR sn=msn
          WAIT " "
          USE careers INDEX careers
          SEEK msn
          REPORT FORM r_career FOR sn=msn
          USE p_eval INDEX p_eval
          SEEK msn
```

```
        REPORT FORM r_eval FOR sn=msn
        WAIT " "
        SELECT 1
        USE rank
        SELECT 2
        USE promote INDEX promote
        JOIN WITH rank  TO temp FOR p_order=A->p_order FIELDS A->ranks,;
           A->tdate,p_order,sn
        USE temp
        INDEX ON p_order TO p_temp
        CLEAR
        @ 5,5 SAY "RANK            PROMOTION            PROMOTION"
        @ 6,5 SAY "                  DATE                 ORDER "
        LIST FOR sn=msn
        CLOSE DATABASES
        WAIT " "
        SELECT 3
        USE awardpun
        SELECT 4
        USE a_p_mn INDEX a_p_mn
        JOIN WITH awardpun TO temp FOR p_order=C->p_order FIELDS C->kind,;
             p_order,C->tdate,sn
        USE temp
        INDEX ON p_order TO p_temp
        CLEAR
        @ 5,5 SAY "  KIND OF AWARD          PERSONNEL ORDER    DATE"
        @ 6,5 SAY "  AND PUNISHMENT                              "
        LIST FOR sn=msn
        WAIT " "
      CASE .NOT. FOUND()
        CLEAR
        @ 17,5 SAY " There is no  " + msn
        @ 24,5 SAY "PRESS ANY KEY TO TRY AGAIN....."
        ? CHR(7)
        WAIT " "
      ENDCASE
ENDDO
*------ When done, return to listing menu.
RETURN
```

5. UPDATE.PRG

```
*************************************************************************
* Module name   :UPDATE.PRG
* Author        :KIM,  SAM NAM
* Date          :27 oct 86
* Purpose       :This is the DRIVER module for the update function.
*                It sets up the main update menu,  accepts input
*                from the user and calls the required modules.
*                When control is returned from the called module,
*                the user is asked if more information is required
*                and the process is repeated,  or control is passed
*                back to the DRIVER module.
* Called by      : DRIVER
* Modules called :MENUSCR,  UPMAIN,  UPEXPERT,  UPEDUCAT,  UPPROMOT,
*                 UP_AW_PU,  MAINTAIN,  UPEVAL,  UPCAREER
* Variables used :
*         Global : i : holds the value of the user input.
*                  today: holds date
*         Local  : none
*************************************************************************
*          Set up loop for presenting menu.
*************************************************************************
CLEAR
DO WHILE .T.
  CLEAR
  DO MENUSCR
  @ 2,14 SAY "U P D A T E   P E R S O N N E L   R E C O R D S"
```

```
@ 6,34 SAY "MAINTAIN"
@ 10,32 SAY "INFORMATION"
@ 8,5 SAY "A. Add new officer members"
@ 10,5 SAY "B. Add expert members"
@ 12,5 SAY "C. Add military education"
@ 13,5 SAY " records"
@ 13,5 SAY "D. Add recent promotion members"
@ 14,5 SAY "E. Add award and punishment records"
@ 9,42 SAY "F. Add performance evaluation"
@ 10,46 SAY " records"
@ 11,42 SAY "G. Add assignment records"
@ 13,42 SAY "I. Change date"
@ 14,42 SAY "X. Return to main menu"
@ 20,0 SAY "DATE          TIME"
@ 20,55 SAY "UPDATED BY"
@ 21,5 SAY today
@ 21,19 SAY TIME()
*@ 21,52 SAY gname
@ 23,10 SAY "[Enter selection (A - I, or X to return to main"
@ 23,57 SAY " menu) : : ]"
DO WHILE .T.
    i=0
    DO WHILE i=0
        i=INKEY()
        @ 21,19 SAY TIME()
        @ 23,65 SAY ""
        IF UPPER(CHR(i))$"ABCDEFGIX"
            EXIT
        ENDIF
        i=0
    ENDDO
    @ 23,65 SAY UPPER(CHR(i))
    IF .NOT. CHR(i)$"Ii"
        EXIT
    ENDIF
    SET COLOR TO W/W
    @ 19,33 SAY "INFORMATION"
    @ 13,42 SAY "I. CHANGE DATE"
    SET COLOR TO W/N
    @ 21,5  GET today
    READ
    @ 21,5 SAY today
    @ 19,33 SAY "INFORMATION"
    @ 13,42 SAY "I. Change date"
    @ 23,65 SAY " "
 ENDDO
 DO CASE
    CASE CHR(i)$ "Xx"
      CLEAR
      DO maintain
      RETURN
    CASE CHR(i)$"Aa"
      DO UPMAIN
    CASE CHR(i)$"Bb"
      DO UPEXPERT
    CASE CHR(i)$"Cc"
      DO UPEDUCAT
    CASE CHR(i)$"Dd"
      DO UPPROMOT
    CASE CHR(i)$"Ee"
      DO UP_AW_PU
    CASE CHR(i)$"Ff"
      DO UPEVAL
    CASE CHR(i)$"Gg"
      DO UPCAREER
  ENDCASE
ENDDO
*---------- when done,return to main menu
RETURN
```

a. MAINTAIN.PRG

```
*************************************************************************
* Module name   :MAINTAIN.PRG
* Author        :KIM,  SAM MAN
* Date          :1 DEC 86
* Purpose       :This is the data file maintenance,  used for
*                updating and editing.
* Called by     : UPDATE,  EDIT
* Modules called : none
*************************************************************************
   USE kim
   DELETE ALL
   PACK
   USE promote INDEX promot
      i = sn
      field = p_order
      DO  WHILE .not. EOF()
          x = i
          y = field
          i = sn
          fieldd = p_order
          IF i = x
              string = SUBSTR(fieldd,1,2)
              fields = SUBSTR(y,1,2)
              IF string > fields
                  field = p_order
              ENDIF
          ENDIF
          IF i <> x
              USE kim
              APPEND BLANK
              REPLACE sns WITH x
              REPLACE porder WITH y
              USE promote INDEX promot
              field =fieldd
          ENDIF
              LOCATE FOR sn = i .AND. p_order = fieldd
              SKIP
      ENDDO
      USE kim
      APPEND BLANK
      REPLACE sns WITH i
      REPLACE porder WITH field
      SELECT 1
      USE kim
      SELECT 2
      USE rank INDEX rank
      JOIN WITH kim TO member FOR p_order = A->porder ;
       FIELDS A->sns,p_order,ranks,tdate
      SELECT 3
      USE main INDEX MAIN
      SELECT 4
      USE member
      JOIN WITH main TO all FOR sns = C->sn ;
        FIELDS ranks,sns,C->name,C->c_type,org_branch,p_order,;
          tdate,C->born_date,C->born_place
      USE all
      REINDEX
      CLOSE DATABASES
      RETURN
```

b. UPMAIN.PRG

```
*************************************************************************
* Module name   :UPMAIN.PRG
* Author        :PARK,  JAE BOCK
* Date          :22 NOV 86
* Purpose       :This is the UPDATE module for updating new member.
*                It sets up the add new officer members menu,
```

```
*               accepts input from the user and calls the
*               required modules. When control is returned
*               from the called module,  the user is asked if more
*               information is required and the process is repeated,
*               or control is passed back to the UPDATE module.
* Called by      : UPDATE
* Modules called :SCREEN1
* Variables used :
*         Global : i : holds the value of the user input.
*                  today: holds date
*          Local  : none
*************************************************************************
*          Set up loop for adding new officer members.
*************************************************************************
USE main Index main
sns = "x"
DO WHILE sns # " "
   CLEAR
   @ 2,1 SAY "Add new officer members"
   @ 2,60 SAY today
   @ 2,70 SAY TIME()
   @ 3,0 SAY ULINE
   ?
   ?
   *------------ Get proposed service number.
   sns = SPACE(8)
   @ 15,5 SAY "Enter service number ( or press RETURN;
   to exit)" GET sns
   READ
   *---------- Check to see if service number already exists.
   sns = LOWER(sns)
   SEEK sns
   DO CASE
      *------- If user did not enter a service number,
      *------- clear the screen and return to UPDATE menu.
      CASE sns = " "
      CLEAR
      *------- If service number already exists,
      *------- notify user and allow another try.
      CASE FOUND()
           @ 20,10 SAY sns + "already exists"
           ? CHR(7)
           WAIT
      *------- If service number not already taken,
      *------- let user add it.
      CASE .NOT. FOUND()
           APPEND BLANK
           REPLACE sn WITH sns
           SET FORMAT TO screen1
           READ
           SET FORMAT TO
      ENDCASE
ENDDO(While user does not enter blank for service number.)
REINDEX
*------------ Return tO UPDATE menu.
RETURN
```

c. **UPEXPERT.PRG**

```
*************************************************************************
* Module name   :UPEXPERT.PRG
* Author        :PARK,  JAE BOCK
* Date          :22 NOV 86
* Purpose       :This is the UPDATE module for adding expert to the EXPERT
*                database file. It sets up the add expert members menu,
*                accepts input from the user and calls the
*                required modules. When control is returned
*                from the called module,  the user is asked if more
*                information is required and the process is repeated,
*                or control is passed back to the UPDATE module.
```

```
* Called by       : UPDATE
* Modules called :none
* Variables used :
*         Global : i : holds the value of the user input.
*                  today: holds date
*         Local  : none
*************************************************************************
*          Set up loop for adding new officer members.
*************************************************************************
sns = "x"
DO WHILE sns # " "
   CLEAR
   @ 2,1 SAY "Add expert members"
   @ 2,60 SAY today
   @ 2,70 SAY TIME()
   @ 3,0 SAY ULINE
   ?
   ?
   *----------- Get proposed service number.
   sns = SPACE(8)
   @ 15,5 SAY "Enter service number ( or press RETURN;
   to exit)" GET sns
   READ
   *---------- Check to see if service number already exists.
   USE main INDEX main
   sns = LOWER(sns)
   SEEK sns
   DO CASE
       *------- If user did not enter a service number,
       *------- clear the screen and return to UPDATE menu.
       CASE sns = " "
       CLEAR
       *------- If service number does not exist,
       *------- notify user and allow another try.
       CASE .NOT. FOUND()
           @ 20,10 SAY sns + "does not exist"
           ? CHR(7)
           WAIT
       *------- If service number already exists in the MAIN file,
       *------- check to see if service number and expert title already
       *------- exists in the EXPERT file.
       CASE  FOUND()
           USE expert INDEX expert
           *------------ Set up loop for adding expert titles.
           ttitle = "x"
           DO WHILE ttitle # " "
               CLEAR
               @ 2,1 SAY "Add expert members"
               @ 2,60 SAY today
               @ 2,70 SAY TIME()
               @ 3,0 SAY ULINE
               ?
               ?
               *----------- Get proposed expert title
               ttitle = SPACE(20)
               @ 15,5 SAY "Enter expert title (or press RETURN to;
   exit)" GET ttitle
               READ
               *-- Check to see if service number already exists.
               SEEK sns
               DO CASE
                   *------- If user did not enter a expert title,
                   *------- clear the screen and return to previous
                   *------- program.
                   CASE ttitle = " "
                     CLEAR
                   *------- If service number does not  exists,
                   *------- add a expert title. .
                   CASE .NOT. FOUND()
```

```
                    APPEND BLANK
                    REPLACE expertitle WITH ttitle
                    REPLACE sn WITH sns
                *------- If service number already exists in the
                *------- EXPERT file, check to see if expert
                *------- title already exists in the EXPERT file.
                CASE FOUND()
                    USE expert INDEX expertitls
                    ttitle = LOWER(ttitle)
                    SEEK ttitle
                    DO CASE
                        CASE FOUND()
                          @ 20,10 SAY ttitle + "already exists"
                          ? CHR(7)
                          WAIT
                          *------- If expert title not already taken,
                          *------- let user add it.
                        CASE .NOT. FOUND()
                          APPEND BLANK
                          REPLACE expertitle WITH ttitle
                          REPLACE sn WITH sns
                    ENDCASE
                ENDCASE
            ENDDO(While user does not enter blank for expert title)
        ENDCASE
        REINDEX
        CLEAR
ENDDO(While user does not enter blank for service number)
*---------- Return to UPDATE menu.
RETURN
```

d. UPEDUCAT.PRG

```
*************************************************************************
* Module name   :UPEDUCAT.PRG
* Author        :KIM,  SAM NAM
* Date          :31 NOV 86
* Purpose       :This is the UPDATE module for adding education results.
*                It quries the user for task selection and calls
*                the correct module to service that request.
*                When control is returned from the calling module,
*                the user is asked if more information is required
*                If yes,  then the process is repeated,  else,  the
*                program terminates.
* Called by      :UPDATE
* Modules called :MENUSCR,  UP_MED1,  UP_M_ED2
* Variables used :
*         Global : i : holds the value of the user input.
*                  today: holds date
*          Local  : none
*************************************************************************
*          Set up loop for presenting menu.
*************************************************************************
CLEAR
  DO WHILE .T.
  * DO WHILE .T. means DO WHILE TRUE i.e. DO FOREVER
  * The DO WHILE will be terminated by an EXIT command
  * Clear the screan and display the main menu
  DO MENUSCR
  @ 2,11 SAY "A D D  M I L I T A R Y  E D C A T I O N  R E S U ;
L T S"
  @ 6,36 SAY "SUBMENU"
  @ 19,33 SAY "INFORMATION"
  @ 10,21 SAY "A. Add military education results"
  @ 11,21 SAY "B. Add personal education results"
  @ 12,21 SAY "C. Change date"
  @ 13,21 SAY "X. Exit"
  @ 20,8 SAY "DATE          TIME"
  @ 20,55 SAY "UPDATED BY"
  @ 21,5 SAY today
```

```
@ 21,19 SAY TIME()
*@ 21,52 SAY gname
@ 23,10 SAY " [ Enter selection ( A - C, or X to Exit ) : : ]"
DO WHILE .T.
    i=0
    DO WHILE i=0
       i=INKEY()
       @ 21,19 SAY TIME()
       @ 23,54 SAY ""
       IF UPPER(CHR(i))$"ABCX"
            EXIT
       ENDIF
       i=0
    ENDDO
    @ 23,54 SAY UPPER(CHR(i))
    IF .NOT. CHR(i)$"Cc"
       EXIT
    ENDIF
    SET COLOR TO N/W
    @ 19,33 SAY "INFORMATION"
    @ 12,21 SAY "C. CHANGE DATE"
    SET COLOR TO W/N
    @ 21,5  GET today
    READ
    @ 21,5 SAY today
    @ 19,33 SAY "INFORMATION"
    @ 12,21 SAY "C. Change date"
    @ 23,54 SAY " "
  ENDDO
  DO CASE
     CASE CHR(i)$ "Xx"
       CLEAR
       RETURN
     CASE CHR(i)$"Aa"
       DO UP_M_ED1
     CASE CHR(i)$"Bb"
       DO UP_M_ED2
   ENDCASE
ENDDO
*---------- when done,return to main menu
RETURN
```

1. UP_M_ED1.PRG

```
*************************************************************************
* Module name   :UP_M_ED1.PRG
* Author        :PARK,  JAE BOCK
* Date          :22 NOV 86
* Purpose       :This is the UPEDCAT module for adding education results.
*                It sets up the add military education result,  accepts
*                input from the user and calls the required modules.
*                When control is returned from the called module,  the
*                user is asked if more information is required and the
*                the process is repeated,  or control is passed back to
*                the UPEDUCAT module.
* Called by       : UPEDUCAT
* Modules called :SCREEN3
* Variables used :
*          Global : i : holds the value of the user input.
*                   today: holds date
*          Local  : none
*************************************************************************
*          Set up loop for adding new officer members.
*************************************************************************
USE m_educat Index m_educat
name= "x"
DO WHILE name # " "
   CLEAR
   @ 2,1 SAY "Add military education results"
   @ 2,60 SAY today
```

```
   @ 2,70 SAY TIME()
   @ 3,0 SAY ULINE
   ?
   ?
   *----------- Get proposed course name.
   name = SPACE(20)
   @ 15,5 SAY "Enter course name ( or press RETURN;
   to exit)" GET name
   READ
   *---------- Check to see if course name already exists.
   name = LOWER(name)
   SEEK name
   DO CASE
      *------- If user did not enter a course name,
      *------- clear the screen and return to UPEDUCAT menu.
      CASE name = " "
      CLEAR
      *------- If course name already exists,
      *------- notify user and allow another try.
      CASE FOUND()
            @ 20,10 SAY name + "already exists"
            ? CHR(7)
            WAIT
      *------- If course name not already taken,
      *------- let user add it.
      CASE .NOT. FOUND()
            APPEND BLANK
            REPLACE cname WITH name
            SET FORMAT TO screen3
            READ
            SET FORMAT TO
      ENDCASE
ENDDO(While user does not enter blank for service number.)
REINDEX
*----------- Return tO UPDATE menu.
RETURN
```

2. UP_M_ED2.PRG

```
*************************************************************************
* Module name   :UP_M_ED2.PRG
* Author        :PARK,  JAE BOCK
* Date          :22 NOV 86
* Purpose       :This is the UPEDUCAT module for updating military
*                education results to the EDUCATMN file.
*                It sets up the add military education results,
*                accepts input from the user and calls the
*                required modules. When control is returned
*                from the called module,   the user is asked if more
*                information is required and the process is repeated,
*                or control is passed back to the UPEDUCAT module.
* Called by      :UPEDUCAT
* Modules called :SCREEN2
* Variables used :
*         Global : i : holds the value of the user input.
*                  today: holds date
*         Local  : none
*************************************************************************
*          Set up loop for adding military education results.
*************************************************************************
USE educatmn INDEX educatmn
name = "x"
sns = "x"
DO WHILE sns # " " .OR. name # " "
  *---------- Find out what service number to add.
   CLEAR
   @ 2,1 SAY "ADD PERSONAL EDUCATION RESULTS"
   @ 2,60 SAY today
   @ 2,70 SAY TIME()
   @ 3,0 SAY ULINE
```

```
?
?
*---------- Get proposed service number and class name.
name = SPACE(20)
sns = SPACE(8)
@ 15,5 SAY "Add for what service number
        " GET sns
READ
@ 17,5 SAY "Add for what class name"
@ 18,7 SAY "( or press RETURN to exit)" GET name
READ
*---------- Try to find that service number.
sns = LOWER(sns)
SEEK sns
DO CASE
    *------- If no service number entered, return to UPDATE menu.
    CASE sns = " "
    CLEAR
    CASE name = " "
    CLEAR
    *------If service number found, try to find that class name
    *------ and add using screen2 format.
    CASE .NOT. FOUND()
      USE educatmn INDEX ccname
          snss = sns + name
          snss = lower(sns)
          SEEK snss
      IF .NOT. FOUND()
      USE educatmn
      APPEND BLANK
      REPLACE sn WITH sns
      REPLACE cname WITH name
      SET FORMAT TO screen2
      READ
      SET FORMAT TO
      ELSE
           @ 5,0 CLEAR
           @ 15,5 SAY name + "   already exists"
           ? CHR(7)
           WAIT
      ENDIF
    *------- Otherwise,warn user and allow another try
    CASE FOUND()
        @ 17,5 SAY "there is no " + sns
        @ 24,5 SAY "Press any key to try again..."
        WAIT " "
    ENDCASE
ENDDO(Continue editing until user requests exit)
*---------- Return tO UPEDCAT menu.
RETURN
```

e. UPPROMOT.PRG

```
*************************************************************************
* Module name   :UPPROMOT.PRG
* Author        :PARK,  JAE BOCK
* Date          :22 NOV 86
* Purpose       :This is the UPDATE module for adding recent
*                promotion members to the PROMOTE database file.
*                It sets up the add recent promotion members menu,
*                accepts input from the user and calls the
*                required modules. When control is returned
*                from the called module,   the user is asked if more
*                information is required and the process is repeated,
*                or control is passed back to the UPDATE module.
* Called by      :UPDATE
* Modules called :MENUSCR,  UPPROMO1,  UPPROMO2
* Variables used :
*     Global : i : holds the value of the user input.
*              today: holds date
```

```
*         Local  : none
*************************************************************************
*         Set up loop for adding promotion officers.
*************************************************************************
CLEAR
   DO WHILE .T.
   * DO WHILE .T. means DO WHILE TRUE i.e. DO FOREVER
   * The DO WHILE will be terminated by an EXIT command
   * Clear the screan and display the main menu
   DO MENUSCR
   @ 2,13 SAY "A D D  R E C E N T  P R O M O T I O N  R E C O R D S"
   @ 6,36 SAY "SUBMENU"
   @ 19,33 SAY "INFORMATION"
   @ 10,21 SAY "A. Add promotion orders"
   @ 11,21 SAY "B. Add personal promotion records"
   @ 12,21 SAY "C. Change date"
   @ 13,21 SAY "X. Exit"
   @ 20,8 SAY "DATE          TIME"
   @ 20,55 SAY "UPDATED BY"
   @ 21,5 SAY today
   @ 21,19 SAY TIME()
   *@ 21,52 SAY gname
   @ 23,10 SAY " [ Enter selection ( A - C, or X to Exit ) : : ]"
   DO WHILE .T.
      i=0
      DO WHILE i=0
         i=INKEY()
         @ 21,19 SAY TIME()
         @ 23,54 SAY ""
         IF UPPER(CHR(i))$"ABCX"
               EXIT
         ENDIF
         i=0
      ENDDO
      @ 23,54 SAY UPPER(CHR(i))
      IF .NOT. CHR(i)$"Cc"
         EXIT
      ENDIF
      SET COLOR TO N/W
      @ 19,33 SAY "INFORMATION"
      @ 12,21 SAY "C. CHANGE DATE"
      SET COLOR TO W/N
      @ 21,5  GET today
      READ
      @ 21,5 SAY today
      @ 19,33 SAY "INFORMATION"
      @ 12,21 SAY "C. Change date"
      @ 23,54 SAY " "
    ENDDO
    DO CASE
       CASE CHR(i)$ "Xx"
          CLEAR
          RETURN
       CASE CHR(i)$"Aa"
          DO UPPROMO1
       CASE CHR(i)$"Bb"
          DO UPPROMO2
     ENDCASE
ENDDO
*----------- when done,return to main menu
RETURN
```

1. UPPROMO1.PRG

```
*************************************************************************
* Module name   :UPPROMO1.PRG
* Author        :PARK,  JAE BOCK
* Date          :22 NOV 86
* Purpose       :This is the UPPROMT module for adding promotion orders
*                to the RANK file. It sets up the add promotion oders,
```

```
*                accepts input from the user and calls the
*                required modules. When control is returned
*                from the called module,  the user is asked if more
*                information is required and the process is repeated,
*                or control is passed back to the UPPROMOT module.
* Called by      : UPPROMOT
* Modules called :SCREEN4
* Variables used :
*          Global : i ; holds the value of the user input.
*                   today: holds date
*          Local  : none
*************************************************************************
*          Set up loop for adding promotion orders.
*************************************************************************
USE rank Index rank
order = "x"
DO WHILE order # " "
   CLEAR
   @ 2,1 SAY "Add recent promotion orders."
   @ 2,60 SAY today
   @ 2,70 SAY TIME()
   @ 3,0 SAY ULINE
   ?
   ?
   *----------- Get proposed promotion order.
   order = SPACE(20)
   @ 15,5 SAY "Enter promotion order ( or press RETURN;
   to exit)" GET order
   READ
   *---------- Check to see if promotion order already exists.
   order = LOWER(order)
   SEEK order
   DO CASE
      *------- If user did not enter a promotion order,
      *------- clear the screen and return to UPPROMOT menu.
      CASE order = " "
      CLEAR
      *------- If promotion order already exists,
      *------- notify user and allow another try.
      CASE FOUND()
           @ 20,10 SAY order + "already exists"
           ? CHR(7)
           WAIT
      *------- If promotion order not already taken,
      *------- let user add it.
      CASE .NOT. FOUND()
           APPEND BLANK
           REPLACE p_order WITH order
           SET FORMAT TO screen4
           READ
           SET FORMAT TO
      ENDCASE
ENDDO(While user does not enter blank for promotion order)
RETINDEX
*----------- Return to UPPROMOT menu.
RETURN
```

2. *UPPROMO2.PRG*

```
*************************************************************************
* Module name   :UPPROMO2.PRG
* Author        :PARK,  JAE BOCK
* Date          :22 NOV 86
* Purpose       :This is the UPPROMOT module for adding personal
*                promotion records to the PROMOTE file.
*                It sets up the add personal promotion record,
*                accepts input from the user and calls the
*                required modules. When control is returned
*                from the called module,  the user is asked if more
*                information is required and the process is repeated
```

```
*                or [illegible] is passed back to the UPPROMOT module.
*  Called by     : [illegible]
*  Calls [called] : none
*  Variables used :
*          Global : i , holds the value of the user input.
*                   today, holds date
*          Local  : none
*****************************************************************
*          Set up loop for adding personal promotion records.
*****************************************************************
USE main INDEX main
sns = " "
DO WHILE sns # " "
  CLEAR
  @ 2,1 SAY "Add personal promotion records"
  @ 2,60 SAY today
  @ 2,70 SAY TIME()
  @ 3,0 SAY ULINE
  ?
  ?
  *---------- Get proposed service number.
  sns = SPACE(9)
  @ 15,5 SAY "Enter service number ( or press RETURN to exit)" GET sns
  READ
  *---------- Check to see if service number already exists.
  sns = LOWER(sns)
  SEEK sns
  DO CASE
     *------- If user did not enter a service number,
     *------- clear the screen and return to UPDATE menu.
     CASE sns = " "
     CLEAR
     *------- If service number does not exist,
     *------- notify user and allow another try.
     CASE .NOT. FOUND()
          @ 20,10 SAY sns + "does not exist"
          ? CHR(7)
          WAIT
     *------- If service number already exists,
     *------- let user add it.
     CASE FOUND()
        USE promote INDEX promote
        *------------ Set up loop for adding personal promotion
        *------------ record.
        orders = "z"
        DO WHILE orders # " "
           CLEAR
           @ 2,1 SAY "Add personal promotion records."
           @ 2,60 SAY today
           @ 2,70 SAY TIME()
           @ 3,0 SAY ULINE
           ?
           ?
           *---------- Get proposed promotion order.
           orders = SPACE(20)
           @ 15,5 SAY "Enter promotion order (or press RETURN to;
exit)"
           @ 16,10 SAY "(e.g --->86-100 ARMY)" GET orders
           READ
           *-- Check to see if service number already exists.
           SEEK sns
           DO CASE
              *------- If user did not enter a promotion order,
              *------- clear the screen and return to previous
              *------- program.
              CASE orders = " "
                 CLEAR
              *------- If service number does not  exists,
              *------- add a personal promotion record to PROMOTE
```

```
                    *------- file.
                    CASE .NOT. FOUND()
                        APPEND BLANK
                        REPLACE p_order WITH orders
                        REPLACE sn WITH sns
                    *------- If service number already exists in the
                    *------- EDUCATHN file, check to see if course
                    *------- name already exists in the EDUCATHN file.
                    CASE FOUND()
                        FIND sns
                        COPY TO temp WHILE sn="sns"
                        USE temp
                        INDEX ON p_order TO temp
                        USE temp INDEX temp
                        orders = LOWER(orders)
                        SEEK orders
                        DO CASE
                            CASE FOUND()
                              @ 20,10 SAY orders + "already exists"
                              ? CHR(7)
                              WAIT
                              *------- If promotion order not already taken,
                              *------- let user add it.
                            CASE .NOT. FOUND()
                              USE promote INDEX promote
                              APPEND BLANK
                              REPLACE p_order WITH orders
                              REPLACE sn WITH sns
                        ENDCASE
                    ENDCASE
                ENDDO(While user does not enter blank for expert title)
            ENDCASE
            REINDEX
            CLEAR
ENDDO(While user does not enter blank for service number.)
*---------- Return to UPDATE menu.
RETURN
```

I. UP_AW_PU.PRG

```
*************************************************************************
* Module name   :UP_AW_PU.PRG
* Author        :KIM,  SAM NAM
* Date          :22 NOV 86
* Purpose       :This is the UPDATE module for adding awarded and
*                punished members to the AWARDPUN,  the A_P_MN,  and
*                the A_P_P database files.
*                It set up the add award and punishment records menu,
*                accepts input from the user and calls the
*                required modules. When control is returned
*                from the called module,  the user is asked if more
*                information is required and the process is repeated,
*                or control is passed back to the UPDATE module.
* Called by       : UPDATE
* Modules called  : MENUSCR,  UPAW_PU1,  UPAW_PU2,  UPAW_PU3
* Variables used :
*          Global : i : holds the value of the user input.
*                   today: holds date
*           Local  : none
*************************************************************************
*      Set up loop for adding award and punishment records.
*************************************************************************
CLEAR
  DO WHILE .T.
  * DO WHILE .T. means DO WHILE TRUE i.e. DO FOREVER
  * The DO WHILE will be terminated by an EXIT command
  * Clear the screan and display the main menu
  DO MENUSCR
  @ 2,11 SAY "A D D  A W A R D  A N D  P U N I S H M E N T  R E ;
C O R D S"
```

```
@ 6,36 SAY "SUBMENU"
@ 19,33 SAY "INFORMATION"
@ 10,21 SAY "A. Add award and punishment points."
@ 11,21 SAY "B. Add personnel order of award and punishment."
@ 12,21 SAY "C. Add personal award and punishment record."
@ 13,21 SAY "D. Change date"
@ 14,21 SAY "X. Exit"
@ 20,8 SAY "DATE          TIME"
@ 20,55 SAY "UPDATED BY"
@ 21,5 SAY today
@ 21,19 SAY TIME()
*@ 21,52 SAY gname
@ 23,10 SAY " [ Enter selection ( A - D, or X to Exit ) : : ]"
DO WHILE .T.
    i=0
    DO WHILE i=0
        i=INKEY()
        @ 21,19 SAY TIME()
        @ 23,54 SAY ""
        IF UPPER(CHR(i))$"ABCDX"
            EXIT
        ENDIF
        i=0
    ENDDO
    @ 23,54 SAY UPPER(CHR(i))
    IF .NOT. CHR(i)$"Dd"
        EXIT
    ENDIF
    SET COLOR TO N/W
    @ 19,33 SAY "INFORMATION"
    @ 13,21 SAY "D. CHANGE DATE"
    SET COLOR TO W/N
    @ 21,5  GET today
    READ
    @ 21,5 SAY today
    @ 19,33 SAY "INFORMATION"
    @ 13,21 SAY "D. Change date"
    @ 23,54 SAY " "
  ENDDO
  DO CASE
     CASE CHR(i)$ "Xx"
        CLEAR
        RETURN
     CASE CHR(i)$"Aa"
        DO UPAW_PU1
     CASE CHR(i)$"Bb"
        DO UPAW_PU2
     CASE CHR(i)$"Cc"
        DO UPAW_PU3
  ENDCASE
ENDDO
*---------- when done,return to main menu
RETURN
```

1. UPAW_PU1.PRG

```
*************************************************************************
* Module name   :UPAW_PU1.PRG
* Author        :KIM,  SAM NAM
* Date          :22 NOV 86
* Purpose       :This is the UP_AW_PU module for adding award and
*                punishment points to the A_P_P file.
*                It sets up the add award and punishment points,
*                accepts input from the user and calls the
*                required modules. When control is returned
*                from the called module,   the user is asked if more
*                information is required and the process is repeated,
*                or control is passed back to the UP_AW_PU module.
* Called by      : UP_AW_PU
* Modules called : SCREEN5
```

```
* Variables used :
*         Global : i ; holds the value of the user input.
*                  today: holds date
*         Local  : none
*************************************************************************
*         Set up loop for adding award and punishment records.
*************************************************************************
USE a_p_p Index a_p_p
kinds = "x"
DO WHILE kinds # " "
   CLEAR
   @ 2,1 SAY "Add award and punishment points."
   @ 2,60 SAY today
   @ 2,70 SAY TIME()
   @ 3,0 SAY ULINE
   ?
   ?
   *----------- Get proposed award and punishment name."
   kinds = SPACE(30)
   @ 15,5 SAY "Enter award and punishment name (press RETURN;
   to exit)" GET kinds
   READ
   *---------- Check to see if award and punishment name already exists.
   kinds = LOWER(kinds)
   SEEK kinds
   DO CASE
      *------- If user did not enter a award and punishment name,
      *------- clear the screen and return to UP_AW_PU menu.
      CASE kinds = " "
      CLEAR
      *------- If award and punishment name already exists,
      *------- notify user and allow another try.
      CASE FOUND()
           @ 20,10 SAY kinds + "already exists"
           ? CHR(7)
           WAIT
      *------- If award and punishment name not already taken,
      *------- let user add it.
      CASE .NOT. FOUND()
           APPEND BLANK
           REPLACE kind WITH kinds
           SET FORMAT TO screen5
           READ
           SET FORMAT TO
      ENDCASE
ENDDO(While user does not enter blank for award and punishment;
name.)
REINDEX
*--- ------- Return tO UP_AW_PU menu.
RETURN
```

2. UPAW_PU2.PRG

```
*************************************************************************
* Module name   :UPAW_PU2.PRG
* Author        :KIM,  SAM NAM
* Date          :22 NOV 86
* Purpose       :This is the UP_AW_PU module for adding the
*                personnel order of award and punishment to the AWARDPUN
*                file. It sets up the add personnel order of award and
*                punishment,  accepts input from the user and calls the
*                required modules. When control is returned
*                from the called module,  the user is asked if more
*                information is required and the process is repeated,
*                or control is passed back to the UP_AW_PU module.
* Called by      : UP_AW_PU
* Modules called :SCREEN6
* Variables used :
*         Global : i : holds the value of the user input.
*                  today: holds date
```

```
*           Local  : None
*****************************************************************
*           Set up loop for adding the personnel order.
*****************************************************************
USE awardpun Index awardpun
award = " "
DO WHILE award # " "
   CLEAR
   @ 2,1 SAY "Add the personnel order of award and punishment."
   @ 2,40 SAY today
   @ 2,70 SAY TIME()
   @ 3,0 SAY WLINE
   ?
   ?
   *------------ Get proposed personnel order.
   award = SPACE(20)
   @ 15,5 SAY "Enter personnel order ( or press RETURN;
   to exit)" GET award
   READ
   *------------ Check to see if personnel order already exists.
   award = LOWER(award)
   SEEK award
   DO CASE
      *-------- If user did not enter a personnel order,
      *-------- clear the screen and return to UP_AW_PU menu.
      CASE award = " "
      CLEAR
      *-------- If personnel order already exists,
      *-------- notify user and allow another try.
      CASE FOUND()
           @ 20,10 SAY award + "already exists"
           ? CHR(7)
           WAIT
      *-------- If personnel order not already taken,
      *-------- let user add it.
      CASE .NOT. FOUND()
           APPEND BLANK
           REPLACE p_order WITH award
           SET FORMAT TO screen6
           READ
           SET FORMAT TO
      ENDCASE
ENDDO(While user does not enter blank for personnel order)
REINDEX
*------------ Return to UP_AW_PU menu.
RETURN
```

3. *UPAW_PU3.PRG*

```
*****************************************************************************
* Module name   :UPAW_PU3.PRG
* Author        :KIM,  SAM NAM
* Date          :22 NOV 86
* Purpose       :This is the UP_AW_PU module for adding personal
*                award and punishment records to the A_P_MN file.
*                It sets up the add personal award and punishment
*                records,  accepts input from the user and calls the
*                required modules. When control is returned
*                from the called module,  the user is asked if more
*                information is required and the process is repeated,
*                or control is passed back to the UP_AW_PU module.
* Called by      : UP_AW_PU
* Modules called : none
* Variables used :
*          Global : i : holds the value of the user input.
*                   today: holds date
*          Local  : none
*****************************************************************************
*           Set up loop for adding military education results.
*****************************************************************************
```

```
USE main INDEX main
sns = "x"
DO WHILE sns # " "
   CLEAR
   @ 2,1 SAY "Add personal award and punishment records."
   @ 2,60 SAY today
   @ 2,70 SAY TIME()
   @ 3,0 SAY ULINE
   ?
   ?
   *---------- Get proposed service number.
   sns = SPACE(8)
   @ 15,5 SAY "Enter service number ( or press RETURN to exit)" GET sns
   READ
   *---------- Check to see if service number already exists.
   sns = LOWER(sns)
   SEEK sns
   DO CASE
      *------- If user did not enter a service number,
      *------- clear the screen and return to UP_AW _PU menu.
      CASE sns = " "
      CLEAR
      *------- If service number does not exist,
      *------- notify user and allow another try.
      CASE .NOT. FOUND()
           @ 20,10 SAY sns + "does not exist"
           ? CHR(7)
           WAIT
      *------- If service number already exists,
      *------- let user add it.
      CASE FOUND()
          USE a_p_mn INDEX a_p_mn
          *----------- Set up loop for adding personal award and
          *----------- punishment records.
          order = "x"
          DO WHILE order # " "
              CLEAR
              @ 2,1 SAY "Add personal award and punishment records."
              @ 2,60 SAY today
              @ 2,70 SAY TIME()
              @ 3,0 SAY ULINE
              ?
              ?
              *---------- Get proposed personnel order.
              order = SPACE(20)
              @ 15,5 SAY "Enter personnel order (or press RETURN to;
exit)"
              @ 16,10 SAY "(e.g --->86-100 ARMY)" GET order
              READ
              *-- Check to see if service number already exists.
              SEEK sns
              DO CASE
                  *------- If user did not enter a personnel order,
                  *------- clear the screen and return to previous
                  *------- program.
                  CASE order = " "
                    CLEAR
                  *------- If service number does not  exists,
                  *------- add a personal award and punishment
                  *------- records to A_P_MN file.
                  CASE .NOT. FOUND()
                      APPEND BLANK
                      REPLACE p_order WITH order
                      REPLACE sn WITH sns
                  *------- If service number already exists in the
                  *------- A_P_MN file, check to see if personnel
                  *------- order already exists in the A_P_MN file.
                  CASE FOUND()
                      FIND &sns
```

```
                    COPY TO temp WHILE sn="&sns"
                    USE temp
                    INDEX ON p_order TO temp
                    USE temp INDEX temp
                    order = LOWER(order)
                    SEEK order
                    DO CASE
                       CASE FOUND()
                          @ 20,10 SAY order + "already exists"
                          ? CHR(7)
                          WAIT
                          *------- If personnel order not already taken,
                          *------- let user add it.
                       CASE .NOT. FOUND()
                          USE a_p_mn INDEX a_p_mn
                          APPEND BLANK
                          REPLACE p_order WITH order
                          REPLACE sn WITH sns
                    ENDCASE
                 ENDCASE
           ENDDO(While user does not enter blank for personnel;
 order.)
        ENDCASE
        CLEAR
ENDDO(While user does not enter blank for service number.)
*---------- Return to UP_AW_PU menu.
RETURN
```

g. UPEVAL.PRG

```
*************************************************************************
* Module name   :UPEVAL.PRG
* Author        :KIM,  SAM NAM
* Date          :22 NOV 86
* Purpose       :This is the UPDATE module for adding personal
*                performance evaluation records to the P-EVAL file.
*                It sets up the add performance evaluation records,
*                accepts input from the user and calls the required
*                modules. When control is returned from the called module,
*                the user is asked if more information is required and
*                the process is repeated,  or control is passed back to
*                the UPDATE module.
* Called by      : UPDATE
* Modules called :SCREEN7
* Variables used :
*         Global : i : holds the value of the user input.
*                  today: holds date
*         Local  : none
*************************************************************************
*      Set up loop for adding personal performance evaluation.
*************************************************************************
USE main INDEX main
sns = "x"
DO WHILE sns # " "
   CLEAR
   @ 2,1 SAY "Add personal performance evaluation records."
   @ 2,60 SAY today
   @ 2,70 SAY TIME()
   @ 3,0 SAY ULINE
   ?
   ?
   *---------- Get proposed service number.
   sns = SPACE(8)
   @ 15,5 SAY "Enter service number ( or press RETURN to exit)" GET sns
   READ
   *---------- Check to see if service number already exists.
   sns = LOWER(sns)
   SEEK sns
   DO CASE
      *------- If user did not enter a service number,
```

```
        *------- clear the screen and return to UPDATE menu.
        CASE sns = " "
        CLEAR
        *------- If service number does not exist,
        *------- notify user and allow another try.
        CASE .NOT. FOUND()
             @ 20,10 SAY sns + "does not exist."
             ? CHR(7)
             WAIT
        *------- If service number already exists,
        *------- let user add it.
        CASE FOUND()
            USE p_eval INDEX p_eval
            *------------ Set up loop for adding personal
            *------------ performance evaluation records.
            rdate = "x"
            DO WHILE rdate # " "
                CLEAR
                @ 2,1 SAY "Add personal performance evaluation;
records."
                @ 2,60 SAY today
                @ 2,70 SAY TIME()
                @ 3,0 SAY ULINE
                ?
                ?
                *----------- Get proposed rating date.
                rdate = SPACE(8)
                @ 15,5 SAY "Enter rating date (or press RETURN to;
 exit)."
                @ 16,10 SAY "(e.g --->MM/DD/YY)" GET rdate
                READ
                *-- Check to see if service number already exists.
                SEEK sns
                DO CASE
                    *------- If user did not enter a rating date,
                    *------- clear the screen and return to previous
                    *------- program.
                    CASE rdate = " "
                      CLEAR
                    *------- If service number does not  exists,add
                    *------- a personal performance evaaluation record
                    *-------    to P_EVAL file.
                    CASE .NOT. FOUND()
                        APPEND BLANK
                        REPLACE ratingdate WITH rdate
                        REPLACE sn WITH sns
                        SET FORMAT TO screen7
                        READ
                        SET FORMAT TO
                    *------- If service number already exists in the
                    *------- P_EVAL file, check to see if rating
                    *------- date already exists in the P_EVAL file.
                    CASE FOUND()
                        FIND &sns
                        COPY TO temp WHILE sn="&sns"
                        USE temp
                        INDEX ON ratingdate TO temp
                        USE temp INDEX temp
                        rdate = LOWER(rdate)
                        SEEK rdate
                        DO CASE
                            CASE FOUND()
                              @ 20,10 SAY rdate + "    " + "already exists."
                              ? CHR(7)
                              WAIT
                              *------- If rating date not already taken,
                              *------- let user add it.
                            CASE .NOT. FOUND()
                              USE p_eval INDEX p_eval
```

```
                        APPEND BLANK
                        REPLACE ratingdate WITH rdate
                        REPLACE sn WITH sns
                        SET FORMAT TO screen7
                        READ
                        SET FORMAT TO
                    ENDCASE
                ENDCASE
            ENDDO(While user does not enter blank for personnel;
 order.)
        ENDCASE
        REINDEX
        CLEAR
ENDDO(While user does not enter blank for service number)
RETURN
```

h. UPCAREER.PRG

```
*************************************************************************
* Module name   :UPCAREER.PRG
* Author        :KIM,  SAM NAM
* Date          :22 NOV 86
* Purpose       :This is the UPDATE module for adding personal
*                military careers to the CAREERS file.
*                It sets up the add assignment records,
*                accepts input from the user and calls the
*                required modules. When control is returned
*                from the called module,  the user is asked if more
*                information is required and the process is repeated,
*                or control is passed back to the UPDATE module.
* Called by      : UPDATE
* Modules called : SCREEN8
* Variables used :
*          Global : i : holds the value of the user input.
*                   today: holds date
*          Local  : none
*************************************************************************
*      Set up loop for adding military career.
*************************************************************************
USE main INDEX main
sns = "x"
DO WHILE sns # " "
   CLEAR
   @ 2,1 SAY "Add assignment records"
   @ 2,60 SAY today
   @ 2,70 SAY TIME()
   @ 3,0 SAY ULINE
   ?
   ?
   *----------- Get proposed service number.
   sns = SPACE(8)
   @ 15,5 SAY "Enter service number ( or press RETURN to exit)" GET sns
   READ
   *---------- Check to see if service number already exists.
   sns = LOWER(sns)
   SEEK sns
   DO CASE
      *------- If user did not enter a service number,
      *------- clear the screen and return to UPDATE menu.
      CASE sns = " "
      CLEAR
      *------- If service number does not exist,
      *------- notify user and allow another try.
      CASE .NOT. FOUND()
           @ 20,10 SAY sns + "does not exist."
           ? CHR(7)
           WAIT
      *------- If service number already exists,
      *------- let user add it.
      CASE FOUND()
```

```
        USE careers INDEX careers
        *------------ Set up loop for adding assignment records.
        order = "x"
        DO WHILE order # " "
            CLEAR
            @ 2,1 SAY "Add assignment records."
            @ 2,60 SAY today
            @ 2,70 SAY TIME()
            @ 3,0 SAY ULINE
            ?
            ?
            *------------ Get proposed personnel order.
            order = SPACE(20)
            @ 15,5 SAY "Enter personnel order (or press RETURN to;
exit)."
            @ 16,10 SAY "(e.g --->85-100 army)" GET order
            READ
            *-- Check to see if service number already exists.
            SEEK sns
            DO CASE
                *------- If user did not enter a personnel order,
                *------- clear the screen and return to previous
                *------- program.
                CASE order = " "
                  CLEAR
                *------- If service number does not  exists,add
                *------- a personal asignment record to CAREERS
                *------- file.
                CASE .NOT. FOUND()
                    APPEND BLANK
                    REPLACE p_order WITH order
                    REPLACE sn WITH sns
                    SET FORMAT TO screen8
                    READ
                    SET FORMAT TO
                *------- If service number already exists in the
                *------- CAREERS file, check to see if personnel
                *------- order already exists in the CAREERS file.
                CASE FOUND()
                    FIND &sns
                    COPY TO temp WHILE sn="&sns"
                    USE temp
                    INDEX ON p_order TO temp
                    USE temp INDEX temp
                    order = LOWER(order)
                    SEEK order
                    DO CASE
                        CASE FOUND()
                          @ 20,10 SAY order + "   " + "already exists."
                          ? CHR(7)
                          WAIT
                          *------- If personnel order not already taken,
                          *------- let user add it.
                        CASE .NOT. FOUND()
                          USE careers INDEX careers
                          APPEND BLANK
                          REPLACE p_order WITH order
                          REPLACE sn WITH sns
                          SET FORMAT TO screen8
                          READ
                          SET FORMAT TO
                    ENDCASE
                ENDCASE
            ENDDO(While user does not enter blank for personnel;
order.)
        ENDCASE
        CLEAR
ENDDO(While user does not enter blank for service number)
*------------ Return to UPdDATE MODULE.
```

```
RETURN
```

6. EDIT.PRG

```
*************************************************************************
* Module name   :EDIT.PRG
* Author        :PARK,  JAE BOCK
* Date          :16 NOV 86
* Purpose       :This is the DRIVER module for the EDIT function
*                It sets up the main edit menu,  accepts input
*                from the user and calls the required modules.
*                When control is returned from the called module,
*                the user is asked if more information is required
*                and the process is repeated or control is passed
*                back to the DRIVER module.
* Called by     : DRIVER
* Modules called :MENUSCR,  EDEXPERT,  EDEDUCAT,  EDPROMOT,  EDAWARD,
*                 EDPUNISH,  EDEVAL,  EDCAREER
* Variables used :
*         Global : i : holds the value of the user input.
*                  today : holds date
*          Local : none
*************************************************************************
*          Set up loop for presenting menu.
*************************************************************************
CLEAR
DO WHILE .T.
   CLEAR
   DO MENUSCR
   @ 2,20 SAY "E D I T   P E R S O N N E L   R E C O R D "
   @ 6,34 SAY "SUBMENU"
   @ 19,33 SAY "INFORMATION"
   @ 9,5 SAY "A. Edit personal personnel record."
   @ 10,5 SAY "B. Edit expert record."
   @ 11,5 SAY "C. Edit military education"
   @ 12,8 SAY " result."
   @ 13,5 SAY "D. Edit promotion record."
   @ 14,5 SAY "E. Edit award and punishment record."
   @ 9,42 SAY "F. Edit performance evaluation"
   @ 10,45 SAY " record."
   @ 11,42 SAY "G. Edit assignment record."
   @ 13,42 SAY "I. Change date"
   @ 14,42 SAY "X. Return to main menu"
   @ 20,8 SAY "DATE          TIME"
   @ 20,55 SAY "UPDATED BY"
   @ 21,5 SAY today
   @ 21,19 SAY TIME()
   *@ 21,52 SAY gname
   @ 23,10 SAY "[Enter selection (A -G, I, or X to return to main"
   @ 23,57 SAY " menu) : : ]"
   DO WHILE .T.
       i=0
       DO WHILE i=0
           i=INKEY()
           @ 21,19 SAY TIME()
           @ 23,65 SAY ""
           IF UPPER(CHR(i))$"ABCDEFGIX"
                 EXIT
           ENDIF
           i=0
       ENDDO
       @ 23,65 SAY UPPER(CHR(i))
       IF .NOT. CHR(i)$"Ii"
           EXIT
       ENDIF
       SET COLOR TO N/W
       @ 19,33 SAY "INFORMATION"
       @ 13,42 SAY "I. CHANGE DATE"
```

```
      SET COLOR TO W/N
      @ 21,5  GET today
      READ
      @ 21,5 SAY today
      @ 19,33 SAY "INFORMATION"
      @ 13,42 SAY "I. Change date"
      @ 23,65 SAY " "
   ENDDO
   DO CASE
      CASE CHR(i)$ "Xx"
        CLEAR
        RETURN
      CASE CHR(i)$"Aa"
        DO EDMAIN
      CASE CHR(i)$"Bb"
        DO EDEXPERT
      CASE CHR(i)$"Cc"
        DO EDEDUCAT
      CASE CHR(i)$"Dd"
        DO EDPROMOT
      CASE CHR(i)$"Ee"
        DO ED_AW_PU
      CASE CHR(i)$"Ff"
        DO EDEVAL
      CASE CHR(i)$"Gg"
        DO EDCAREER
   ENDCASE
ENDDO
*---------- when done,  return to main menu.
RETURN
```

a. EDMAIN.PRG

```
*************************************************************************
* Module name   :EDMAIN.PRG
* Author        :PARK,  JAE BOCK
* Date          :22 NOV 86
* Purpose       :This is the EDIT module for editing personnel record.
*                It sets up the edit prsonal personnel record,
*                accepts input from the user and calls the
*                required modules. When control is returned
*                from the called module,  the user is asked if more
*                information is required and the process is repeated,
*                or control is passed back to the EDIT module.
* Called by       : EDIT
* Modules called :SCREEN11
* Variables used :
*          Global : i : holds the value of the user input.
*                   today: holds date
*          Local  : none
*************************************************************************
*          Set up loop for editing personnel record.
*************************************************************************
USE main Index main
sns = "x"
DO WHILE sns # " "
   *---------- Find out what service number to edit.
   CLEAR
   @ 2,1 SAY "EDIT PERSONNEL RECORD"
   @ 2,60 SAY today
   @ 2,70 SAY TIME()
   @ 3,0 SAY ULINE
   ?
   ?
   *---------- Get proposed service number.
   sns = SPACE(8)
   @ 15,5 SAY "Edit for what service number ( or press RETURN;
   to exit)" GET sns
   READ
   *---------- Try to find that service number.
```

```
   sns = LOWER(sns)
   SEEK sns
   DO CASE
      *------- If no service number entered, return to EDIT menu.
      CASE sns = " "
      CLEAR
      *------- If service number found, edit using SCREEN1 format.
      CASE FOUND()
         SET FORMAT TO screen11
         READ
         SET FORMAT TO
      *------- Otherwise,  warn user and allow another try.
      CASE .NOT. FOUND()
          @ 17,5 SAY "There is no  " + sns + "number."
          @ 24,5 SAY "Press any key to try again..."
          WAIT " "
      ENDCASE
ENDDO(Continue editing until user requests exit)
*---------- Return to EDIT menu.
RETURN
```

b. EDEXPERT.PRG

```
*************************************************************************
* Module name   :EDEXPERT.PRG
* Author        :KIM,  SAM NAM
* Date          :22 NOV 86
* Purpose       :This is the EDIT module for editing  an expert record.
*                It sets up the edit expert record,  accepts input from
*                the user and calls the  required modules.
*                When control is returned from the called module,
*                the user is asked if more information is required and
*                the process is repeated,  or control is passed back
*                to the EDIT module.
* Called by       : EDIT
* Modules called  : SCREEN9
* Variables used  :
*          Global : i : holds the value of the user input.
*                   today: holds date
*           Local : none
*************************************************************************
*          Set up loop for editing an expert record.
*************************************************************************
USE expert INDEX expert
title = "x"
sns = "x"
DO WHILE sns # " " .AND. title # " "
  *---------- Find out what service number to edit.
   CLEAR
   @ 2,1 SAY "EDIT EXPERT RECORD"
   @ 2,60 SAY today
   @ 2,70 SAY TIME()
   @ 3,0 SAY ULINE
   ?
   ?
   *---------- Get proposed service number.
   title = SPACE(20)
   sns = SPACE(8)
   @ 15,5 SAY "Edit for what service number ( or press RETURN;
   to exit)" GET sns
   READ
   @ 15,5 SAY "Edit for what expert title   ( or press RETURN;
   to exit)" GET title
   READ
   *---------- Try to find that service number.
   sns = LOWER(sns)
   SEEK sns
   DO CASE
      *------- If no service number entered, return to EDIT menu.
      CASE sns = " "
```

```
[illegible]
[illegible] title = " "
[illegible]
*------- If service number found, edit using screen9 format.
[illegible]()
   [illegible] export [illegible] exportitle
      [illegible] = [illegible] + title
      [illegible] = [illegible](name)
      [illegible] name
   [illegible]()
   [illegible] FORMAT TO screen9
   [illegible]
   [illegible] FORMAT TO
   ELSE
      @ 5,0 CLEAR
      @ 15,5 SAY title + "   Not found"
      ? CHR(7)
      WAIT
   ENDIF
*------- Otherwise,warn user and allow another try.
CASE .not. FOUND()
   @ 17,5 SAY "There is no  " + name + " number."
   @ 20,5 SAY "Press any key to try again..."
   WAIT " "
ENDCASE
ENDDO(Continue editing until user requests exit.)
*---------- Return to EDIT menu.
RETURN
   c. EDMEDUCAT.PRG

*************************************************************************
* Module name   :EDMEDUCAT.PRG
* Author        :FROM, JAE ROCK
* Date          :22 NOV 88
* Purpose       :This is the EDIT module for editing military education
*                record. It sets up the edit military education result,
*                accepts input from the user and calls the
*                required modules. When control is returned
*                from the called module, the user is asked if more
*                information is required and the process is repeated,
*                or control is passed back to the EDIT module.
* Called by     : EDIT
* Modules called : none
* Variables used :
*         Global : i : holds the value of the user input.
*                  today: holds date
*         Local  : none
*************************************************************************
*          Set up loop for editing education result.
*************************************************************************
CLEAR
   DO WHILE .T.
   * DO WHILE .T. means DO WHILE TRUE i.e. Do forever
   * The DO WHILE will be terminated by an EXIT command.
   * Clear the screen and display the main menu.
   DO MENUSCR
   @ 2,12 SAY "E D I T  M I L I T A R Y  E D U C A T I O N "
   @ 6,36 SAY "SUMMARY"
   @ 10,33 SAY "INFORMATION"
   @ 10,21 SAY "A.Edit military education result."
   @ 11,21 SAY "B.Edit personal education result."
   @ 12,21 SAY "C. Change date"
   @ 13,21 SAY "X. Exit"
   @ 20,8 SAY "DATE          TIME"
   @ 20,55 SAY "UPDATED BY"
   @ 21,5 SAY today
   @ 21,19 SAY TIME()
   *@ 21,52 SAY gname
   @ 23,10 SAY " [ Enter selection ( A - C, or X to Exit ) : : ]"
```

```
DO WHILE .T.
   IF
   DO WHILE i=0
      DO WHILE(
      @ [illegible] SAY TIME()
      @ [illegible] SAY IF
      IF [illegible](i))$"ABCR"
         EXIT
      ENDIF
      IF
   ENDDO
   @ 23,54 SAY UPPER(CHR(i))
   IF .NOT. CHR(i)$"Cc"
      EXIT
   ENDIF
   SET COLOR TO N/W
   @ 19,33 SAY "INFORMATION"
   @ 12,21 SAY "C. CHANGE DATE"
   SET COLOR TO W/N
   @ 21,5  GET today
   READ
   @ 21,5 SAY today
   @ 19,33 SAY "INFORMATION"
   @ 12,21 SAY "C. Change date"
   @ 23,54 SAY " "
 ENDDO
 DO CASE
   CASE CHR(i)$ "Xx"
      CLEAR
      RETURN
   CASE CHR(i)$"Aa"
      DO ED_M_ED1
   CASE CHR(i)$"Bb"
      DO ED_M_ED2
 ENDCASE
ENDDO
*---------- Return to EDIT menu.
RETURN
```

1. ED_M_ED1.PRG

```
*************************************************************************
* Module name   :ED_M_ED1.PRG
* Author        :KIM,   SAM MAM
* Date          :22 NOV 86
* Purpose       :This is the EDEDCAT module for editing military
*                education results. It sets up the edit military education
*                result,  accepts input from the user and calls the
*                required modules. When control is returned
*                from the called module,  the user is asked if more
*                information is required and the process is repeated,
*                or control is passed back to the EDEDUCAT module.
* Called by      : EDEDUCAT
* Modules called :SCREEN31
* Variables used :
*          Global : i : holds the value of the user input.
*                   today: holds date
*          Local  : none
*************************************************************************
*          Set up loop for editing personnel record.
*************************************************************************
USE m_educat Index m_educat
class = "x"
DO WHILE class # " "
   *---------- Find out what class name to edit.
   CLEAR
   @ 2,1 SAY "EDIT MILITARY EDUCATION RESULTS"
   @ 2,60 SAY today
   @ 2,70 SAY TIME()
   @ 3,0 SAY ULINE
```

```
?
?
*---------- Get proposed class name.
class = SPACE(20)
@ 15,5 SAY "Edit for what class name ( or press RETURN;
to exit)" GET class
READ
*---------- Try to find that class name.
class = LOWER(class)
SEEK class
DO CASE
    *------- If no class name entered, return to EDCAT menu.
    CASE class = " "
    CLEAR
    *------- If class name found, edit using SCREEN1 format.
    CASE FOUND()
      SET FORMAT TO screen31
      READ
      SET FORMAT TO
    *------- Otherwise,warn user and allow another try.
    CASE .NOT. FOUND()
       @ 17,5 SAY "There is no  " + class
       @ 24,5 SAY "Press any key to try again..."
       WAIT " "
    ENDCASE
ENDDO(Continue editing until user requests exit.)
*---------- Return to EDCAT menu.
RETURN
```

2. *ED_M_ED2.PRG*

```
*************************************************************************
* Module name   :ED_M_ED2.PRG
* Author        :KIM,  SAM NAM
* Date          :22 NOV 86
* Purpose       :This is the EDEDCAT module for editing  personal
*                education results.
*                It sets up the edit personal education result,
*                accepts input from the user and calls the
*                required modules. When control is returned
*                from the called module,  the user is asked if more
*                information is required and the process is repeated,
*                or control is passed back to the EDEDCAT module.
* Called by      : EDEDCAT
* Modules called : SCREEN21
* Variables used :
*         Global : i : holds the value of the user input.
*                  today: holds date
*          Local  : none
*************************************************************************
*          Set up loop for editing personal education result.
*************************************************************************
USE educatmn INDEX educatmn
name = "x"
sns = "x"
DO WHILE sns # " " .OR. name # " "
  *---------- Find out what service number to edit.
   CLEAR
   @ 2,1 SAY "EDIT PERSONAL EDUCATION RESULTS"
   @ 2,60 SAY today
   @ 2,70 SAY TIME()
   @ 3,0 SAY ULINE
   ?
   ?
   *---------- Get proposed service number and class name.
   name = SPACE(20)
   sns = SPACE(6)
   @ 15,5 SAY "Edit for what service number                    ;
            " GET sns
   READ
```

```
   @ 17,5 SAY "Edit for what class name"
   @ 18,7 SAY "( or press RETURN to exit)" GET name
   READ
   *---------- Try to find that service number.
   sns = LOWER(sns)
   SEEK sns
   DO CASE
      *------- If no service number entered, return to EDIT menu.
      CASE sns = " "
      CLEAR
      CASE name = " "
      CLEAR
      *------If service number found, try to find that class name
      *------ and edit using screen21 format.
      CASE FOUND()
        USE educatmn INDEX ccname
            sns = sns + name
            sns = lower(sns)
            SEEK sns
        IF FOUND()
        SET FORMAT TO screen21
        READ
        SET FORMAT TO
        ELSE
             @ 5,0 CLEAR
             @ 15,5 SAY name + "   Not found"
             ? CHR(7)
             WAIT
        ENDIF
      *------- Otherwise,warn user and allow another try.
      CASE .NOT. FOUND()
         @ 17,5 SAY "There is no  " + sns
         @ 24,5 SAY "Press any key to try again..."
         WAIT " "
      ENDCASE
ENDDO(Continue editing until user requests exit.)
*---------- Return to EDEDCAT menu.
RETURN
```

d. EDPROMOT.PRG

```
*************************************************************************
* Module name   :EDPROMOT.PRG
* Author        :KIM,  SAM NAM
* Date          :22 NOV 86
* Purpose       :This is the EDIT module for editing promotion
*                records. It sets up the edit promotion record,
*                accepts input from the user and calls the
*                required modules. When control is returned
*                from the called module,  the user is asked if more
*                information is required and the process is repeated,
*                or control is passed back to the EDIT module.
* Called by      : EDIT
* Modules called : none
* Variables used :
*          Global : i : holds the value of the user input.
*                   today: holds date
*          Local  : none
*************************************************************************
*          Set up loop for editing education result.
*************************************************************************
CLEAR
   DO WHILE .T.
   * DO WHILE .T. means DO WHILE TRUE i.e. Do forever.
   * The DO WHILE will be terminated by an EXIT command.
   * Clear the screan and display the main menu.
   DO MENUSCR
   @ 2,12 SAY "E D I T  P R M O T I O N  R E C O R D "
   @ 6,36 SAY "SUBMENU"
   @ 19,33 SAY "INFORMATION"
```

```
@ 10,21 SAY "A.Edit promotion record."
@ 11,21 SAY "B.Edit personal promotion record."
@ 12,21 SAY "C.Change date"
@ 13,21 SAY "X.Exit"
@ 20,3 SAY "DATE                TIME"
@ 20,35 SAY "UPDATED BY"
@ 21,3 SAY today
@ 21,19 SAY TIME()
*@ 21,32 SAY cname
@ 23,10 SAY " [ Enter selection ( A - C, or X to Exit ) : : ]"
DO WHILE .T.
   i=0
   DO WHILE i=0
      i=INKEY()
      @ 21,19 SAY TIME()
      @ 23,54 SAY ""
      IF UPPER(CHR(i))$"ABCX"
         EXIT
      ENDIF
      i=0
   ENDDO
   @ 23,54 SAY UPPER(CHR(i))
   IF .NOT. CHR(i)$"Cc"
      EXIT
   ENDIF
   SET COLOR TO N/W
   @ 19,33 SAY "INFORMATION"
   @ 12,21 SAY "C. CHANGE DATE"
   SET COLOR TO W/N
   @ 21,5  GET today
   READ
   @ 21,5 SAY today
   @ 19,33 SAY "INFORMATION"
   @ 12,21 SAY "C. Change date"
   @ 23,54 SAY " "
ENDDO
DO CASE
   CASE CHR(i)$ "Xx"
      CLEAR
      RETURN
   CASE CHR(i)$"Aa"
      DO EDPROMO1
   CASE CHR(i)$"Bb"
      DO EDPROMO2
ENDCASE
ENDDO
*---------- Return to EDIT menu.
RETURN
```

1. EDPROMO1.PRG

```
*************************************************************************
* Module name   :EDPROMO1.PRG
* Author        :KIM,  SAM MAN
* Date          :22 NOV 86
* Purpose       :This is the EDPROMOT module for editing promotion
*                record. It sets up the edit promotion record,
*                accepts input from the user and calls the
*                required modules. When control is returned
*                from the called module,  the user is asked if more
*                information is required and the process is repeated,
*                or control is passed back to the EDPROMOT module.
* Called by      : EDPROMOT
* Modules called :SCREEN41
* Variables used :
*          Global : i : holds the value of the user input.
*                   today: holds date
*          Local  : none
*************************************************************************
*          Set up loop for editing PROMOTION record.
```

```
[illegible]
[illegible] = " "
DO WHILE order # " "
   *---------- Find out what promotion order to edit.
   CLEAR
   @ 2,1 SAY "EDIT PROMOTION RECORDS"
   @ 2,60 SAY today
   @ 1,70 SAY TIME()
   @ 3,0 SAY [illegible]
   ?
   ?
   *---------- Get proposed promotion order.
   order = SPACE(20)
   @ 15,5 SAY "Edit for what promotion order ( or press RETURN;
   to exit)" GET order
   READ
   *---------- Try to find that promotion order.
   order = LOWER(order)
   SEEK order
   DO CASE
      *------- If no promotion order entered, return to EDPROMOT menu.
      CASE order = " "
      CLEAR
      *------- If promotion order found, edit using SCREEN4 format.
      CASE FOUND()
        SET FORMAT TO screen41
        READ
        SET FORMAT TO
      *------- Otherwise,warn user and allow another try.
      CASE .NOT. FOUND()
        CLEAR
        @ 17,5 SAY "There is no  " + order
        @ 24,5 SAY "Press any key to try again..."
        WAIT " "
      ENDCASE
ENDDO(Continue editing until user requests exit.)
*---------- Return to EDPROMOT menu.
RETURN
```

2. *EDPROMO2.PRG*

```
*************************************************************************
* Module name   :EDPROMO2.PRG
* Author        :KIM,  SAM NAM
* Date          :22 NOV 86
* Purpose       :This is the EDPROMOT module for editing  personal
*                promotion record.It sets up the edit personal promotion
*                record,  accepts input from the user and calls the
*                required modules. When control is returned
*                from the called module,  the user is asked if more
*                information is required and the process is repeated,
*                or control is passed back to the EDPROMOT module.
* Called by      : EDPROMOT
* Modules called :
* Variables used :
*          Global : i : holds the value of the user input.
*                   today: holds date.
*          Local  : none
*************************************************************************
*          Set up loop for editing personal promotion record.
*************************************************************************
USE promote INDEX promote
order = "x"
sns = "x"
DO WHILE sns # " " .OR. order # " "
  *---------- Find out what service number to edit.
   CLEAR
   @ 2,1 SAY "EDIT PERSONAL PROMOTION RECORDS"
   @ 2,60 SAY today
```

```
@ 3,70 SAY TIME()
@ 5,5 SAY CLOCK
*
*
*---------- Get proposed service number and promotion order.
order = SPACE(20)
sns = SPACE(8)
@ 15,5 SAY "Edit for what service number .                    ;
        " GET sns
READ
@ 17,5 SAY "Edit for what promotion order"
@ 18,7 SAY " ( or press RETURN to exit)" GET order
READ
*---------- Try to find that service number.
sns = LOWER(sns)
SEEK sns
DO CASE
    *------- If no service number entered, return to EDPROMOT menu.
    CASE sns = " "
    CLEAR
    CASE order = " "
    CLEAR
    *------If service number found, try to find that promotion
    *------ order and edit it.
    CASE FOUND()
      USE promote INDEX p_order
        sns = sns + order
        sns = lower(sns)
        SEEK sns
      IF FOUND()
        SET FORMAT TO screen10
        READ
        SET FORMAT TO
      ELSE
        @ 5,0 CLEAR
        @ 15,5 SAY order + "   Not found"
        ? CHR(7)
        WAIT
      ENDIF
    *------- Otherwise,warn user and allow another try.
    CASE .NOT. FOUND()
      CLEAR
      @ 17,5 SAY "There is no  " + sns
      @ 24,5 SAY "Press any key to try again..."
      WAIT " "
    ENDCASE
ENDDO(Continue editing until user requests exit.)
*---------- Return to EDPROMOT menu.
RETURN
```

e. ED_AW_PU.PRG

```
*************************************************************************
* Module name   :ED_AW_PU.PRG
* Author        :KIM,  SAM NAM
* Date          :22 NOV 86
* Purpose       :This is the EDIT module for editing award and punishment
*                record. It sets up the edit award and punishment record,
*                accepts input from the user and calls the
*                required modules. When control is returned
*                from the called module,  the user is asked if more
*                information is required and the process is repeated,
*                or control is passed back to the EDIT module.
* Called by      : EDIT
* Modules called :
* Variables used :
*          Global : i : holds the value of the user input.
*                   today: holds date
*           Local  : None
*************************************************************************
```

```
*           Set up loop for editing award and punishment record.
*************************************************************************
CLEAR
   DO WHILE .T.
   * DO WHILE .T. means DO WHILE TRUE i.e. DO FOREVER
   * The DO WHILE will be terminated by an EXIT command
   * Clear the screan and display the main menu
   DO MENUSCR
   @ 2,12 SAY "EDIT  AWARD  PUNISHMENT  RECORD "
   @ 6,36 SAY "SUBMENU"
   @ 19,33 SAY "INFORMATION"
   @ 10,21 SAY "A.Edit award and punishment points."
   @ 11,21 SAY "B.Edit award and punishment record."
   @ 12,21 SAY "C.Edit personal award and punishment record."
   @ 13,21 SAY "D.Change date."
   @ 14,21 SAY "X.Exit"
   @ 20,8 SAY "DATE          TIME"
   @ 20,55 SAY "UPDATED BY"
   @ 21,5 SAY today
   @ 21,19 SAY TIME()
   *@ 21,52 SAY gname
   @ 23,10 SAY " [ Enter selection ( A - D, or X to Exit ) : : ]"
   DO WHILE .T.
      i=0
      DO WHILE i=0
         i=INKEY()
         @ 21,19 SAY TIME()
         @ 23,54 SAY ""
         IF UPPER(CHR(i))$"ABCDX"
               EXIT
         ENDIF
         i=0
      ENDDO
      @ 23,54 SAY UPPER(CHR(i))
      IF .NOT. CHR(i)$"Dd"
         EXIT
      ENDIF
      SET COLOR TO N/W
      @ 19,33 SAY "INFORMATION"
      @ 13,21 SAY "D. CHANGE DATE"
      SET COLOR TO W/N
      @ 21,5  GET today
      READ
      @ 21,5 SAY today
      @ 19,33 SAY "INFORMATION"
      @ 13,21 SAY "D. Change date"
      @ 23,54 SAY " "
    ENDDO
    DO CASE
       CASE CHR(i)$ "Xx"
          CLEAR
          RETURN
       CASE CHR(i)$"Aa"
          DO EDAW_PU1
       CASE CHR(i)$"Bb"
          DO EDAW_PU2
       CASE CHR(i)$"Cc"
          DO EDAW_PU3
     ENDCASE
ENDDO
*---------- Return tO EDIT menu.
RETURN
```

1. EDAW_PU1.PRG

```
*************************************************************************
* Module name   :EDAW_PU1.PRG
* Author        :KIM,  SAM NAM
* Date          :22 NOV 86
* Purpose       :This is the ED_AW_PU module for editing award and
```

```
*               punishment ponits. It sets up the edit award and
*               punishment points,  accepts input from the user and
*               calls quired modules. When control is returned
*               from the called module,   the user is asked if more
*               information is required and the process is repeated,
*               or control is passed back to the ED_AW_PU module.
* Called by      : ED_AW_PU
* Modules called : SCREEN51
* Variables used :
*         Global : i : holds the value of the user input.
*                  today: holds date
*         Local  : none
*************************************************************************
*          Set up loop for editing award and punishment point.
*************************************************************************
USE a_p_p Index a_p_p
award = "x"
DO WHILE award # " "
   *---------- Find out what award and punishment point to edit.
   CLEAR
   @ 2,1 SAY "EDIT AWARD AND PUNISHMENT POINTS"
   @ 2,60 SAY today
   @ 2,70 SAY TIME()
   @ 3,0 SAY ULINE
   ?
   ?
   *---------- Get proposed type of award and punishment.
   award = SPACE(30)
   @ 15,5 SAY "Edit for what kind of award and punishment"
   @ 16,7 SAY "( or press RETURN to exit)" GET award
   READ
   *---------- Try to find that kind of award and punishment.
   award = LOWER(award)
   SEEK award
   DO CASE
      *------- If no kind of awardand punishment entered,
      *-------  return to ED_AW_PU menu.
      CASE award = " "
      CLEAR
      *------- If kind of award and punishment found,
      *-------   edit using SCREEN5 format.
      CASE FOUND()
         SET FORMAT TO screen51
         READ
         SET FORMAT TO
      *------- Otherwise,warn user and allow another try.
      CASE .NOT. FOUND()
         CLEAR
         @ 17,5 SAY "There is no  " + award
         @ 24,5 SAY "Press any key to try again..."
         WAIT " "
      ENDCASE
ENDDO(Continue editing until user requests exit.)
*---------- Return to ED_AW_PU menu.
RETURN
```

2. *EDAW_PU2.PRG*

```
*************************************************************************
* Module name   :EDAW_PU2.PRG
* Author        :KIM,  SAM NAM
* Date          :22 NOV 86
* Purpose       :This is the ED_AW_PU module for editing award and
*               punishment records.
*               It sets up the edit award and punishment record,
*               accepts input from the user and calls the
*               required modules. When control is returned
*               from the called module,  the user is asked if more
*               information is required and the process is repeated,
*               or control is passed back to the ED_AW_PU module.
```

```
* Called by       : ED_AW_PU
* Modules called :SCREEN61
* Variables used :
*         Global : i : holds the value of the user input.
*                  today: holds date
*         Local  : none
*************************************************************************
*          Set up loop for editing award and punishment point.
*************************************************************************
USE awardpun Index awardpun
award = "x"
DO WHILE award # " "
   *---------- Find out what award and punishment record to edit.
   CLEAR
   @ 2,1 SAY "EDIT AWARD AND PUNISHMENT RECORDS"
   @ 2,60 SAY today
   @ 2,70 SAY TIME()
   @ 3,0 SAY ULINE
   ?
   ?
   *----------- Get proposed  PRSONNEL ORDER.
   award = SPACE(20)
   @ 15,5 SAY "Edit for what personnel order"
   @ 16,7 SAY "( or press RETURN to exit)" GET award
   READ
   *---------- Try to find that personnel order.
   award = LOWER(award)
   SEEK award
   DO CASE
      *------- If no personnel order entered,return to ED_AW_PU menu.
      CASE award = " "
      CLEAR
      *------- If personnel order found, edit using SCREEN61 format.
      CASE FOUND()
         SET FORMAT TO screen61
         READ
         SET FORMAT TO
      *------- Otherwise,  warn user and allow another try.
      CASE .NOT. FOUND()
         CLEAR
         @ 17,5 SAY "There is no  " + award
         @ 24,5 SAY "Press any key to try again..."
         WAIT " "
      ENDCASE
ENDDO(Continue editing until user requests exit.)
*---------- Return tO ED_AW_PU menu.
RETURN
```

3. EDAW_PU3.PRG

```
*************************************************************************
* Module name   :EDAW_PU3.PRG
* Author        :KIM,  SAM NAM
* Date          :22 NOV 86
* Purpose       :This is the ED_AW_PU module for editing  personal
*                award and punishment records.
*                It sets up the edit personal award and punishment record,
*                accepts input from the user and calls the
*                required modules. When control is returned
*                from the called module,  the user is asked if more
*                information is required and the process is repeated,
*                or control is passed back to the ED_AW_PU module.
* Called by       : ED_AW_PU
* Modules called : none
* Variables used :
*         Global : i : holds the value of the user input.
*                  today: holds date
*         Local  : none
*************************************************************************
*    Set up loop for editing personal award and punishment record.
```

```
*************************************************************************
USE a_p_mn INDEX a_p_mn
order = "x"
sns = "x"
DO WHILE sns # " " .OR. order # " "
  *---------- Find out what service number to edit.
   CLEAR
   @ 2,1 SAY "EDIT PERSONAL AWARD AND PUNISHMENT RECORDS."
   @ 2,60 SAY today
   @ 2,70 SAY TIME()
   @ 3,0 SAY ULINE
   ?
   ?
   *---------- Get proposed service number and personnel order.
   order = SPACE(20)
   sns = SPACE(8)
   @ 15,5 SAY "Edit for what service number                    ;
          " GET sns
   READ
   @ 17,5 SAY "Edit for what personnel order "
   @ 18,7 SAY "( or press RETURN to exit)                " GET order
   READ
   *---------- Try to find that service number.
   sns = LOWER(sns)
   SEEK sns
   DO CASE
      *------- If no service number entered, return to ED_AW_PU menu.
      CASE sns = " "
          CLEAR
      CASE order = " "
         CLEAR
      *------If service number found, try to find that personnel
      *------ order and edit it.
      CASE FOUND()
        USE a_p_mn INDEX a_p_mns
            sns = sns + order
            sns = lower(sns)
            SEEK sns
        IF FOUND()
            SET FORMAT TO screen12
            READ
            SET FORMAT TO
        ELSE
             @ 5,0 CLEAR
             @ 15,5 SAY order + "   Not found"
             ? CHR(7)
             WAIT
        ENDIF
      *------- Otherwise,warn user and allow another try.
      CASE .NOT. FOUND()
        CLEAR
        @ 17,5 SAY "There is no  " + sns
        @ 24,5 SAY "Press any key to try again..."
        WAIT " "
      ENDCASE
ENDDO(Continue editing until user requests exit.)
*---------- Return to ED_AW_PU menu.
RETURN
```

f. EDEVAL.PRG

```
*************************************************************************
* Module name   :EDEVAL.PRG
* Author        :KIM,  SAM NAM
* Date          :22 NOV 86
* Purpose       :This is the EDIT module for editing performance
*                evaluation records.
*                It sets up the edit performance evaluation record,
*                accepts input from the user and calls the
*                required modules. When control is returned
```

```
*                from the called module,  the user is asked if more
*                information is required and the process is repeated,
*                or control is passed back to the EDIT module.
* Called by       : EDIT
* Modules called  : SCREEN71
* Variables used  :
*          Global : i : holds the value of the user input.
*                   today: holds date
*           Local  : none
*************************************************************************
*        Set up loop for editing performance evaluation record.
*************************************************************************
USE p_eval INDEX p_eval
rdate = "x"
sns = "x"
DO WHILE sns # " " .OR. rdate # " "
 *---------- Find out what service number to edit.
   CLEAR
   @ 2,1 SAY "EDIT PERSONAL PROMOTION RECORDS"
   @ 2,60 SAY today
   @ 2,70 SAY TIME()
   @ 3,0 SAY ULINE
   ?
   ?
   *----------- Get proposed service number and promotion order.
   rdate = SPACE(8)
   sns = SPACE(8)
   @ 15,5 SAY "Edit for what service number .                ;
     " GET sns
   READ
   @ 17,5 SAY "Edit for what rating date (e.g-----> 03/10/86"
   @ 18,7 SAY " ( or press RETURN to exit)              " GET rdate
   READ
   *---------- Try to find that service number.
   sns = LOWER(sns)
   SEEK sns
   DO CASE
       *------- If no service number entered, return to EDIT menu.
       CASE sns = " "
       CLEAR
       CASE rdate = " "
       CLEAR
       *------If service number found, try to find that rating
       *------ date and edit it.
       CASE FOUND()
          USE p_eval INDEX rating
              sns = sns + rdate
              sns = lower(sns)
              SEEK sns
          IF FOUND()
          SET FORMAT TO screen71
          READ
          SET FORMAT TO
          ELSE
               @ 5,0 CLEAR
               @ 15,5 SAY rdate + "   Not found"
               ? CHR(7)
               WAIT
          ENDIF
       *------- Otherwise,warn user and allow another try.
       CASE .NOT. FOUND()
          CLEAR
          @ 17,5 SAY "There is no  " + sns
          @ 24,5 SAY "Press any key to try again..."
          WAIT " "
       ENDCASE
ENDDO(Continue editing until user requests exit.)
*---------- Return to EDIT menu.
RETURN
```

g. EDCAREER.PRG

```
*************************************************************************
* Module name   :EDCAREER.PRG
* Author        :KIM,  SAM NAM
* Date          :22 NOV 86
* Purpose       :This is the EDIT module for editing  personal
*                assignment records.
*                It sets up the edit personal assignment record,
*                accepts input from the user and calls the
*                required modules. When control is returned
*                from the called module,  the user is asked if more
*                information is required and the process is repeated,
*                or control is passed back to the EDIT module.
* Called by      : EDIT
* Modules called : SCREEN81
* Variables used :
*         Global : i : holds the value of the user input.
*                  today: holds date
*         Local  : none
*************************************************************************
*          Set up loop for editing assignment record.
*************************************************************************
USE careers INDEX careers
order = "x"
sns = "x"
DO WHILE sns # " " .OR. order # " "
  *---------- Find out what service number to edit.
   CLEAR
   @ 2,1 SAY "EDIT PERSONAL ASSIGNMENT RECORDS"
   @ 2,60 SAY today
   @ 2,70 SAY TIME()
   @ 3,0 SAY ULINE
   ?
   ?
   *---------- Get proposed service number and personnel order.
   order = SPACE(20)
   sns = SPACE(8)
   @ 15,5 SAY "Edit for what service number .            ;
      " GET sns
   READ
   @ 17,5 SAY "Edit for what personnel order"
   @ 18,10 SAY " ( or press RETURN to exit)   " GET order
   READ
   *---------- Try to find that service number.
   sns = LOWER(sns)
   SEEK sns
   DO CASE
      *------- If no service number entered, return to EDIT menu.
      CASE sns = " "
      CLEAR
      CASE order = " "
      CLEAR
      *------If service number found, try to find that personnel
      *------ order and edit it.
      CASE FOUND()
        USE careers INDEX career
            sns = sns + order
            sns = lower(sns)
            SEEK sns
        IF FOUND()
        SET FORMAT TO screen81
        READ
        SET FORMAT TO
        ELSE
             @ 5,0 CLEAR
             @ 15,5 SAY order + "   Not found"
             ? CHR(7)
             WAIT
```

```
      ENDIF
      *-------- Otherwise,warn user and allow another try.
      CASE .NOT. FOUND()
        CLEAR
        @ 17,5 SAY "There is no  " + sns
        @ 24,5 SAY "Press any key to try again..."
        WAIT " "
      ENDCASE
ENDDO(Continue editing until user requests exit.)
*---------- Return tO EDIT menu.
RETURN
```

7. DICT.PRG

```
*************************************************************************
*  MODULE NAME   :  DICT.PRG
*  AUTHOR        :  KIM,  SAM NAM
*  DATE : 31 NOV 1986
*  PURPOSE : This program allows the personnel officer to
*          obtain information about the data in the database.
*  CALLED BY :  DRIVER
*  MODULES CALLED :  A problem developed as we used the DRIVER
*                    to call subprograms(too many files open).
*                    This program is the main driver for the
*                    dictionary requests.
*  VARIABLES USED:
*       GLOBAL :  i : holds the value of the user input.
*                 today : holds date
*       LOCAL  :  none
*************************************************************************
CLEAR
   DO WHILE .T.
    CLEAR
   * DO WHILE .T. means DO WHILE TRUE i.e. do forever.
   * The DO WHILE will be terminated by an EXIT command.
   * Clear the screan and display the main menu.
   DO MENUSCR
   @ 2,20 SAY "DATA DICTIONARY INQUIRIES "
   @ 6,36 SAY "SUBMENU"
   @ 19,33 SAY "INFORMATION"
   @ 10,9 SAY "A.What is ??? and how do I enter it into the"
   @ 10,53 SAY " computer?"
   @ 11,9 SAY "B.What type of information does a particular"
   @ 11,55 SAY " file contain?"
   @ 12,9 SAY "C.What file contains a particular variable?"
   @ 13,9 SAY "D.Dictionary information."
   @ 14,9 SAY "E.Change date"
   @ 15,9 SAY "X. Exit"
   @ 20,8 SAY "DATE          TIME"
   @ 20,55 SAY "UPDATED BY"
   @ 21,5 SAY today
   @ 21,19 SAY TIME()
   *@ 21,52 SAY gname
   @ 23,10 SAY " [ Enter selection ( A - E, or X to Exit ) : : ]"
   DO WHILE .T.
       i=0
       DO WHILE i=0
           i=INKEY()
           @ 21,19 SAY TIME()
           @ 23,54 SAY ""
           IF UPPER(CHR(i))$"ABCDEX"
                 EXIT
           ENDIF
           i=0
       ENDDO
       @ 23,54 SAY UPPER(CHR(i))
       IF .NOT. CHR(i)$"Ee"
           EXIT
```

```
[illegible]
DO CASE
   CASE [illegible](1)$ "[illegible]"
      [illegible]
   CASE [illegible](1)$"[illegible]"
      DO [illegible]
   CASE [illegible](1)$"[illegible]"
      DO [illegible]
   CASE [illegible](1)$"[illegible]"
      DO [illegible]
   CASE [illegible](1)$"[illegible]"
      DO [illegible]
   ENDCASE
ENDDO
*----------- when done,return to main menu
RETURN
```

6. DICT1.PRG

```
***********************************************************************
*       MODULE NAME     :  DICT1.PRG
*       AUTHOR          :  [illegible]
*       DATE            :  [illegible]
*       PURPOSE:     This  module allows a user to discover what an
*                    element  is,  what it's allowable values are,
*                    and how to enter it into the computer.
*       CALLED BY :    [illegible].PRG
*       MODULES CALLED : none
*       VARIABLES USED:
*            LOCAL  :  varname, printout
*            GLOBAL : none
***********************************************************************
          CLEAR GETS
          CLEAR
          STORE "                    " TO varname
          STORE " " TO printout
          USE element
          * ---- Displays the menu for making the element name
          *----- request
          @ 2,6 SAY "This portion of the data dictionary;
inquiries allows yo"
          @ 2,61 SAY "u to"
          @ 3,6 SAY "ask questions about elements  and how;
they are entered"
          @ 3,61 SAY "into"
          @ 4,6 SAY "the computer.  For example, you know the;
element  name of"
          @ 5,6 SAY  "somthing  is    SN       , but you don't ;
know what it means"
          @ 6,6 SAY "or how to enter it.  You would type the;
element name at the"
          @ 7,6 SAY "Promt and would get the full name    ,;
allowable values, and"
          @ 8,6 SAY "How to enter it."
          @ 9,6 SAY "     If you're not sure how to enter it,"
          @ 10,6 SAY "      enter what you do know"
          @ 14,10
          ACCEPT "Please enter the element  name-- " TO varname
```

```
@ 20,1 SAY "Do you want a printout? y OR n"
@ 20,36 GET printout
READ
varname = LOWER(varname)
IF varname = "help"
   [illegible]
ENDIF
IF printout = "y"
   SET PRINT ON
ENDIF
*----- This section displays all elements with the required
*------ element name
        DISPLAY ALL "Element ID ",elementid for;
 elementid=varname off
        DISPLAY ALL "Full ID",fullid FOR elementid = varname off
        DISPLAY ALL "Type ", type FOR elementid = varname off
        DISPLAY ALL "COMMENTS--", comments FOR elementid =;
varname off
        SET PRINT OFF
        WAIT
RETURN
```

B. DICT2.PRG

```
*******************************************************************
*     MODULE NAME   :  DICT2.PRG
*     AUTHOR        :  [illegible],  [illegible]
*     DATE :   31 MAY 1988
*     PURPOSE:   Allows the user to find out how commonly used files
*                are structured.
*     CALLED BY :   DICT.PRG
*     MODULES CALLED : none
*     VARIABLES USED:
*          LOCAL :   FILENAME,   PRINTOUT
*          GLOBAL : i : holds value of user input.
*                   today: holds date
*******************************************************************
*          Set up loop for presenting menu.
*******************************************************************
CLEAR
DO WHILE .T.
   CLEAR
   DO [illegible]
   @ 2,14 SAY "DATA DICTIONARY ABOUT DATA FILE STRUCTURE"
   @ 3,[illegible] SAY "[illegible]"
   @ 4,[illegible] SAY "INFORMATION"
   @ 6,[illegible] SAY "A. Main file"
   @ [illegible] SAY "B. Military education file"
   @ [illegible] SAY "C. Personal education file"
   @ [illegible] SAY "D. Rank file"
   @ [illegible] SAY "E. Promote file"
   @ [illegible] SAY "F. Award and punishment records"
   @ [illegible] SAY "G. Personal award and punishment"
                     " file"
   @ [illegible] SAY "H. Award and punishment point file"
   @ [illegible] SAY "I. Report file"
   @ [illegible] SAY "J. Performance evaluation file"
   @ [illegible] SAY "K. Military careers file "
   @ [illegible] SAY "L. File file"
   @ [illegible] SAY "M. User file"
   @ [illegible] SAY "N. Change date"
   @ [illegible] SAY "X. Return to main menu"
   @ 20,[illegible] SAY "DATE
                          TIME"
   @ 20,[illegible] SAY "UPDATED BY"
   @ 21,[illegible] SAY today
   @ 21,[illegible] SAY TIME()
   @ 21,12 SAY [illegible]
   @ 23,[illegible] SAY "[Enter selection (A - N, or X to return to main"
   @ 24,[illegible] SAY " menu) : : ]"
   DO WHILE .T.
```

```
[illegible]
```

```
        CASE CMD(1)$"Ii"
           USE export
           LIST struc
           EXIT
        CASE CMD(1)$"Jj"
           USE p_proj
           LIST struc
           EXIT
        CASE CMD(1)$"Kk"
           USE careers
           LIST struc
           EXIT
        CASE CMD(1)$"Ll"
           USE file
           LIST struc
           EXIT
        CASE CMD(1)$"Mm"
           USE user
           LIST struc
           EXIT
     ENDCASE
     SET PRINT OFF
ENDDO
*------------ when done,return to main menu
     c. DICT3.PRG

*************************************************************************
*      MODULE NAME   :  DICT3.PRG
*      AUTHOR        :  KIM,  SAM MAM
*      DATE  :  31 NOV 1986
*      PURPOSE:  Allows the user to see what elements are used
*                in other files, programs, and etc.
*      CALLED BY :  DICT.PRG
*      MODULES CALLED :none
*      VARIABLES USED:
*           LOCAL :  vername, printout
*           GLOBAL : none
*************************************************************************
CLEAR GETS
STORE
STORE
STORE " " to printout
USE contains
*------- Displays the menu then requests the variable name.
     @ 2,10 SAY "This portion of the data dictionary allows you to make"
     @ 3,10 SAY "queries about where certain pieces of information are"
     @ 4,10 SAY "used throughout system.  For example, if you wanted"
     @ 5,10 SAY "to know where P_CHARGE was used (perhaps to assess th"
     @ 6,10 SAY "e"
     @ 7,10 SAY "possible impact of transitioning to the 15 digit code),"
     @ 8,10 SAY "you would enter the variable name at the prompt and the"
     @ 9,10 SAY "information you would receive would be listed in the fo"
     @ 10,10 SAY "rm:"
     @ 11,10 SAY "name of the place where used   name of the file,program"
     @ 12,10 SAY "etc."
     @ 13,10 SAY "which uses it"
     @ 14,10 SAY "such as                MAIN    file"
     @ 15,10 SAY "driver   program"
     @ 17,5 SAY "If you don't know the exact name,"
     @ 18,5 SAY "enter what you do know, in all lower   letters"
     @ 20,0
     ACCEPT "Please enter the variable name-- " TO vername
     @ 22,5 SAY "Do you want a printout? y or n"
     @ 22,36 GET printout
     READ
     IF vername = "help"
        HELP
     ENDIF
     IF printout = "y"
```

```
        SET PRINT ON
    EJECT
        SET HEADING OFF
*--- Displays all the names and entity types for the required
*--- variable name.
    DISPLAY ALL einame,eitype FOR e2name = varname OFF
    [illegible]
    DISPLAY ALL id1,type1 for id2 = varname OFF
    [illegible]
    SET PRINT OFF
    [illegible]
RETURN
```

d. DICT4.PRG

```
*****************************************************************
*     MODULE NAME   :  DICT4.PRG
*     AUTHOR        :  KIM,  SAM NAM
*     DATE : 31 NOV 1988
*     PURPOSE:  Allows  the  user to exit the   data dictionary
*     queries  and gives the more experienced user some information
*     on how to use other features of the dictionary.
*     CALLED BY :  DICT.PRG
*     MODULES CALLED : none
*     VARIABLES USED,
*          LOCAL : none
*          GLOBAL : none
****************************************************************
CLEAR
@ 1,7 SAY "Before you leave the data dictionary, you should know t"
@ 1,62 SAY "hat other"
@ 2,7 SAY "questions can be answered about this SYSTEM database"
@ 2,60 SAY "system"
@ 3,7 SAY "using the DBMS III+ query language.  The previous querie"
@ 3,63 SAY "s were"
@ 4,7 SAY "designed to answer typical questions a primary user of"
@ 4,62 SAY "the system"
@ 5,7 SAY "might have while using the system. For the individual w"
@ 5,62 SAY "ith a"
@ [illegible],7 SAY "knowledge of DBMS III+, information exists about"
@ [illegible],7 SAY "SYSTEM-- The PERSONNEL MANAGEMENT SYSTEM"
@ [illegible],7 SAY "PROGRAM-- Programs used in the PERSONNEL MANAGEMENT SYSTEM"
@ 10,7 SAY "CONTAINS--  Information about where entities are contai"
@ 10,62 SAY "ned"
@ 11,15 SAY "in other  entities"
@ [illegible],7 SAY "PROCESSES--  Information about where entities act upon"
@ [illegible],62 SAY "another entity."
@ [illegible],7 SAY "ELEMENT-- Descriptions of the instances of data such as"
@ [illegible],63 SAY "service"
@ [illegible],15 SAY "number"
@ [illegible],7 SAY "FILE--  Describes the files used in this system "
@ [illegible],7 SAY "USER--  Describes users of in this system."
WAIT
RETURN
```

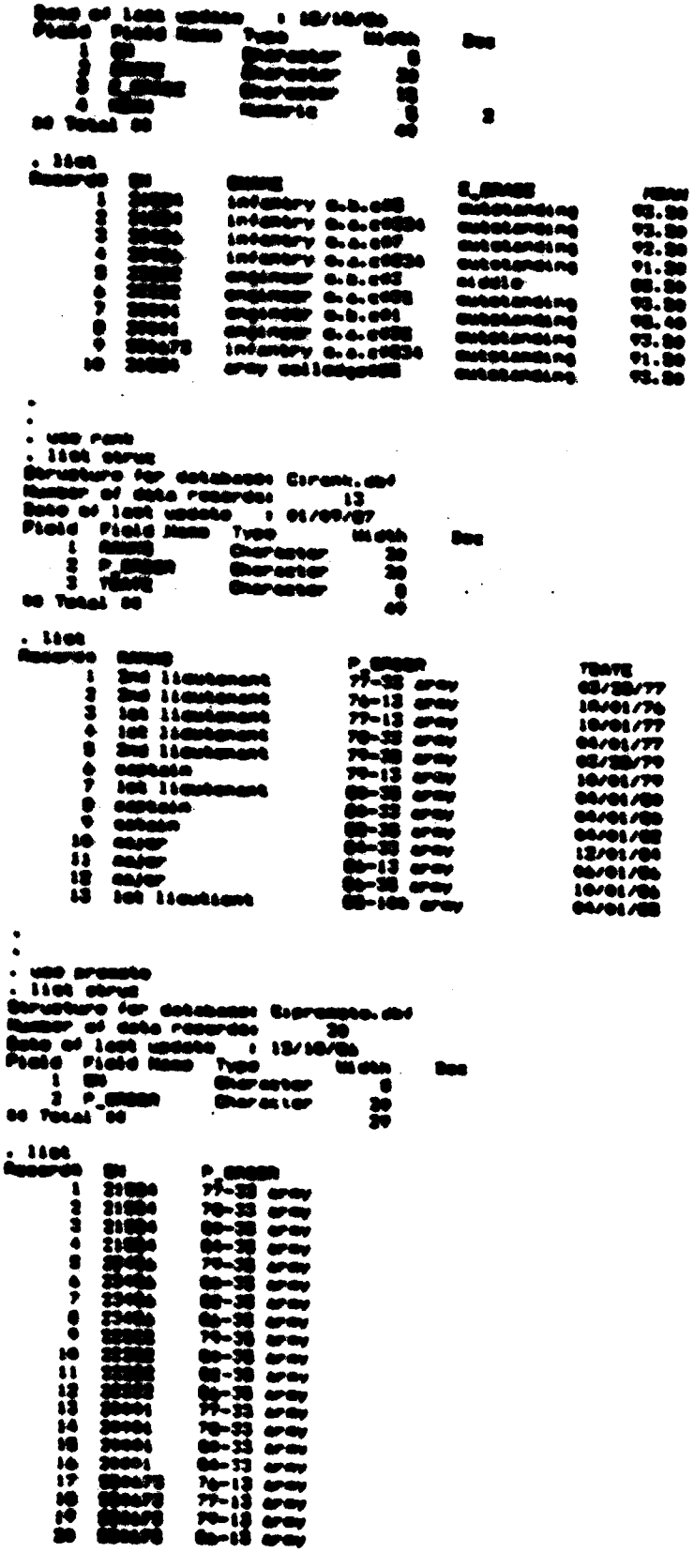
APPENDIX C
DATA DUMP

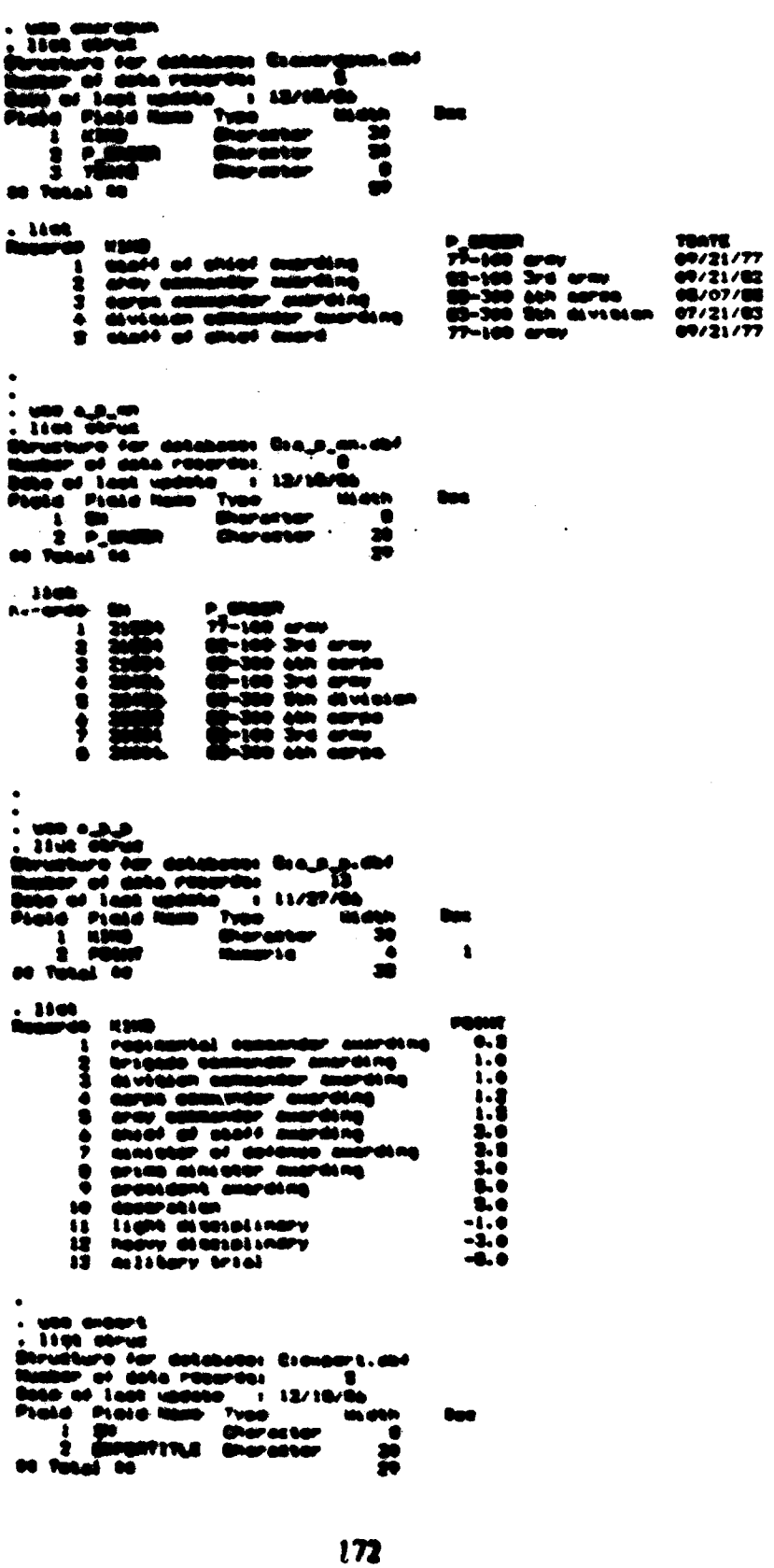
```
. use main
. list struc
Structure for database: C:main.dbf
Number of data records:        6
Date of last update    : 01/31/87
Field  Field Name  Type       Width     Dec
    1  SN          Character        8
    2  NAME        Character       25
    3  ORG_BRANCH  Character       20
    4  C_TYPE      Character       10
    5  BORN_DATE   Character        8
    6  BORN_PLACE  Character       15
** Total **                       87

. list
Record#  SN       NAME                    ORG_BRANCH          C_TYPE     BORN
_DATE BORN_PLACE
      1  21884    kim,soo nam             infantry            K.M.A833   07/2
1/34  chungnam do
      2  23456    park,joo baek           infantry            K.M.A835   08/2
1/56  seoul city
      3  22222    kang,seon mo            engineer            K.M.A838   09/2
1/56  seoul city
      4  20001    jung,joo ho             infantry            K.M.A833   06/2
1/58  pusan city
      5  880673   joo,doo joon            infantry            3K.M.A813  06/2
1/58  kyounggee do
      6  22345

.
.
. use a_educa
. list struc
Structure for database: C:a_educa.dbf
Number of data records:        6
Date of last update    : 12/02/86
Field  Field Name  Type       Width     Dec
    1  ENAME       Character       20
    2  CLASS_SIZE  Numeric          4
    3  START_DATE  Character        8
    4  END_DATE    Character        8
    5  SNAME       Character       30
    6  CLASS_MEAN  Numeric          5       2
** Total **                       76

. list
Record#  ENAME                CLASS_SIZE START_DATE END_DATE SNAME
      CLASS_MEAN
      1  infantry o.b.c05            200 04/01/77   07/01/77 army infantry schoo
l             80.00
      2  engineer o.b.c03             35 04/01/79   07/01/79 army engineer schoo
l             79.00
      3  infantry o.b.c0834          188 06/01/82   11/20/82 army infantry schoo
l             82.00
      4  engineer o.b.c038            60 07/21/84   12/01/84 army engineer schoo
l             80.00
      5  infantry o.b.c07            200 04/01/79   07/01/79 army infantry schoo
l             88.00
      6  engineer o.b.c01             40 04/01/77   07/01/77 army engineer schoo
l             78.00
```





```
. list
Records:  [redacted]  DESIGNTITLE
     1  [redacted]  0.0.0
     2  [redacted]  0.0.0
     3  [redacted]  civil engineer
     4  [redacted]  0.0.0
     5  [redacted]  0.0.0

.
.
. use p_eval
. list struc
Structure for database: [redacted]p_eval.dbf
Number of data records:        4
Date of last update    : 12/10/86
Field  Field Name  Type       Width    Dec
    1  [redacted]  Character        8
    2  [redacted]  Character        2
    3  [redacted]  Character        8
** Total **                       19

. list
Records:  [redacted]  GRADE RATINGDATE
     1  [redacted]  [redacted]  02/10/79
     2  [redacted]  [redacted]  02/10/79
     3  [redacted]  [redacted]  02/10/80
     4  [redacted]  [redacted]  02/10/81

.
.

. use career
. list struc
Structure for database: [redacted]career.dbf
Number of data records:        5
Date of last update    : 01/31/87
Field  Field Name  Type       Width    Dec
    1  [redacted]  Character        8
    2  [redacted]  Character        8
    3  [redacted]  Character        8
    4  [redacted]  Character        7
    5  [redacted]  Character       30
** Total **                       62

. list
Records:  [redacted]  [redacted]  [redacted]  [redacted]  [redacted]
     1  [redacted]  [redacted]  [redacted]  [redacted]  [redacted]
     2  [redacted]  [redacted]  [redacted]  [redacted]  [redacted]
     3  [redacted]  [redacted]  [redacted]  [redacted]  [redacted]
     4  [redacted]  [redacted]  [redacted]  [redacted]  [redacted]
     5  [redacted]  [redacted]  [redacted]  [redacted]  [redacted]
     6  [redacted]  [redacted]             [redacted]  [redacted]
     7  [redacted]  [redacted]  [redacted]  [redacted]  [redacted]
     8  [redacted]  [redacted]  [redacted]  [redacted]  [redacted]

.
.
. use unit
. list struc
Structure for database: [redacted]unit.dbf
Number of data records:        8
Date of last update    : 01/31/87
Field  Field Name  Type       Width    Dec
    1  [redacted]  Character        7
    2  [redacted]  Character       20
    3  [redacted]  Character       22
** Total **                       50

. list
Records:  [redacted]  [redacted]  [redacted]
     1  [redacted]  [redacted]  [redacted]
     2  [redacted]  [redacted]  [redacted]
     3  [redacted]  [redacted]  [redacted]
     4  [redacted]  [redacted]  [redacted]
     5  [redacted]  [redacted]  [redacted]
     6  [redacted]  [redacted]  [redacted]
     7  [redacted]  [redacted]  [redacted]
     8  [redacted]  [redacted]  [redacted]
```

APPENDIX D
QUERY SAMPLE

- **Service number :** 21554
- **Personnel order or promotion order :** 86-100 army
- **Course name :** infantry o.a.c#234
- **Promotion year :** 85
- **Rank :** major
- **Expert title :** c.p.a
- **Award and punishment name :** army commander awarding
- **Evaluation date :** 03/10/85

LIST OF REFERENCES

1. Cruse, Donald P., *Personnel, The Management of Human Resources*, Wadsworth Publishing Company INC.,1966.

2. Dunnette, Marvin D., *Personnel Selection and Placement*, Wadsworth Publishing INC.,1966.

3. Bassett, Glenn A., and Weatherbee, Harvard Y., *Personnel Systems and Data Management*, American Management Association INC.,1971.

4. Walsh, Myles E., *Information Management Systems: Virtual Storage*, Reston Publishing Company INC.,1979.

5. Martin James, *Computer Data Base Organization*, Prentice-Hall,1977.

6. Ullman, Jeffery D., *Principles of Database System* . Computer Science Press INC.,1980.

7. Kroenke, David M., *Database Processing: Fundamentals, Design, Implementation*, Science Research Associates INC., 1983.

8. Deen S. M., *Fundamentals of Data Base Systems*, Hayden Book company INC.,1973.

9. Teory, Toby J. and Fry, James P., *Design of database structures*, Prentice -Hall INC.,1982.

10. Tsichritzis, Dionysios C. and Lochovsky, Frederick H., *Database Management System*, Academic Press INC.,1977.

11. Duyn, Van V., *Developing a data dictionary system*, Prentice-Hall INC .1982.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Computer Technology Programs,Code 37 Naval Postgraduate School Monterey, California 93943-5000	1
4.	Department Chairman,Code 54 Dept. of Administrative Science Monterey, California 93943-5000	1
5.	Professor Norman R. Lyons, Code 54Lb Naval Postgraduate School Monterey, California 93943-5000	1
6.	Professor Richard A. McGonigal, Code 54Mb Naval Postgraduate School Monterey, California 93943-5000	1
7.	Central Computer Center Army Headquarters, [illegible] Seoul, Republic of Korea	1
8.	Central Computer Center Air Force Headquarters Seoul, Republic of Korea	1
9.	Central Computer Center Department of Defence Seoul, Republic of Korea	1
10	Park, Jae Rock Chunnam Yokpo Si Sanyang 3 Dong 9T2B 9 Seoul, Republic of Korea	1
11	Kim, Soon Nam Chunnam Hampyung Gun Eomda Myun Songro Ri 149 Seoul, Republic of Korea	1

END

7-87

DTIC