A MICROCOMPUTER DATA BASE MANAGEMENT SYSTEM

CONVERSION OF

THE RISS DATA BASE MANAGEMENT SYSTEM

FOR MICROCOMPUTER USE

BY

KATHLEEN AWAI, B.Sc.

A Report
Submitted to the School of Graduate Studies
in Partial Fulfilment of the Requirements
for the Degree

Master of Science (Computation)

McMaster University

December 1980

TO JESSE

MASTER OF SCIENCE (1980) McMaster University
(Computation) Hamilton, Ontario

TITLE: Conversion of the RISS Database System for Microcomputer Use

AUTHOR: Kathleen Awai, B.Sc. (University of West Indies)

SUPERVISOR: Professor N. P. Archer

NUMBER OF PAGES: vi, 122

ABSTRACT

The RISS (Relational Inquiry and Storage System) data base management system was first implemented at Forest Hospital in Des Plaines, Illinois. It was originally written in BASIC-PLUS to run under the RSTS/E operating system on a DEC PDP-11 minicomputer. The RISS system used the relational data base structure because of its basic simplicity and because of the ease with which new relations may be added to the data base without disturbing existing applications.

The aim of this project was to convert the existing RISS software from BASIC-PLUS to CBASIC-2 to run on the Dynabyte microcomputer under the CP/M operating system, for use in the McMaster University Faculty of Business. A simple application program was also developed to demonstrate the converted RISS system.

ACKNOWLEDGEMENTS

First of all I want to thank Dr. N. P. Archer for assisting me in my choice of a project and then for supervising this project. His recurrent and enthusiastic encouragement throughout the course of this work has been of paramount importance to its progress. I am grateful too for his helpful comments on the initial draft.

I would also like to direct special thanks to Dr. R. Welke for taking an active interest in the project and for assisting with any difficulties I experienced with the computer hardware.

I am especially grateful to the Government of Trinidad and Tobago, without whose sponsorship, my studies in Canada would not have been possible.

Finally, I would like to thank my mother for her devotion to my two children at a time when they most needed it.

TABLE OF CONTENTS

		Page
CHAPTER I	INTRODUCTION	
1.1	The Data Base Approach	4
1.2	Advantages of Data Base Systems	6
1.3	The Three Data Models	8
1.4	Choice of DBMS	11
1.5	The Relational Data Model	13
1.6	Choice of Language	15
1.7	A Microcomputer Data Base Management System	17
CHAPTER II	STRUCTURE OF RISS DBMS FOR MICROCOMPUTERS	
2.1	Logical Structure	19
2.2	Physical Structure	20
2.3	Data Manipulation	23
2.3.1	The Naive-user Interface Level	24
2.3.2	The Application Program Interface Level	42
2.4	Integrity and Privacy Controls	48
CHAPTER III	DIFFERENCES BETWEEN THE PUBLISHED VERSION AND THE MICROCOMPUTER CBASIC-2 VERSION OF THE RISS DATA BASE SYSTEM	
3.1	Physical Data Organization	50
3.2	Differences in Operating System	58
3.3	Application Program Interface	59
3.4	Naive-user Interface	60
3.5	Program Chaining	61
3.6	Error Recovery	63
3.7	Differences in CBASIC-2 Statements	64

CHAPTER IV SAMPLE DATA BASE APPLICATION

4.1	Evaluating the Need for an Application Program	72
4.2	Purpose of the Application Program	73
4.3	Logical Data Base Description	73
4.4	Description of Data Base Files	75
4.5	Building the Data Base Files	81
4.6	Loading the Data Base	82
4.7	Application Program Description	83
4.8	Privacy Controls	93

CHAPTER V CONCLUSION

5.1	Disadvantages of the Relational Model	96
5.2	Limitations of CBASIC-2	97
5.3	Limitations of Microcomputers	98
5.4	Future Enhancements	99
5.5	Concluding Remarks	101

APPENDIX A Sample Dialogue for Naive-user Interface of Microcomputer DBMS 102

APPENDIX B Listing of Sample Application Program 116

BIBLIOGRAPHY 121

CHAPTER I

INTRODUCTION

In recent years, there has been a substantial growth in the use of Data Base Management Systems by business computer users. This has been brought about largely by a reduction in the cost of computer hardware, and the development of a variety of suitable data base software packages, all combining to make this approach so cost effective in managing business data.

Concurrently, with the increased business use of small computer systems, there is now a developing need for data base management systems with these small computers. This project was directed towards adapting RISS (Relational Inquiry and Storage System [1]), which was developed for use on a minicomputer, to run on a microcomputer.

The RISS data base management system was developed at Forest Hospital, Illinois , on a DEC PDP-11 minicomputer, to run under the control of the RSTS/E operating system. RSTS/E is a time sharing system designed to accomodate large numbers of interactive users. The programs were written in BASIC-PLUS, a version of BASIC designed for

RSTS/E and therefore offering many special features not available in other versions of BASIC.

In RISS the relational model of data was adopted, as opposed to hierarchical and network data base structures, because it permits the user to view data as being stored in two dimensional tables, a logical design easily grasped by relatively naive users. Also, the relational approach was preferred because of the ease with which new relations may be added to the data base without disturbing existing applications.

Some of the applications that have been implemented at Forest Hospital are payroll, accounts receivable, inventory, test scoring, research studies and statistical tabulations. The RISS data base management system has been in use for the past five years, requiring only minor modifications to take advantage of improvements in the RSTS/E operating system and the BASIC-PLUS language. The programs have been well tested and are sufficiently bug free to serve as a reliable vehicle for a data base management system.

The primary objective of this project was to convert the existing RISS software to run on the Dynabyte microcomputer under the CP/M operating system for use in the McMaster University Faculty of Business. The programs were converted to a widely used commercial BASIC, CBASIC Version 2 (CBASIC-2). CBASIC-2 will run under the

control of any CP/M operating system.

Although a conversion of this sort would represent little difficulty if the original system were written in a standardized language, BASIC is not standardized. Hence, there was a great deal of difficulty in adapting operations written in the original BASIC-PLUS to CBASIC-2, especially when these operations involved file manipulations and commands related to time sharing functions.

A secondary objective of this work was to develop an application program to demonstrate the use of the converted RISS data base management system in creating, accessing and maintaining a data base.

The remainder of this chapter explains the data base concept and introduces the relational data model on which RISS is based. A discussion on the choice of a data model and the subsequent choice of a data base management system is also included. Chapter II describes the general structure of the converted data base management system and provides a detailed outline of its two interface levels, the 'naive'-user interface and the application-user or programming user interface.

Chapter III points out the differences between the published version of RISS [1] and the adapted CBASIC version. In Chapter IV, a sample data base application is given to demonstrate the converted RISS

software. Chapter V concludes with an evaluation of the system developed and suggests future improvements that could be made to the converted system.

1.1 The Data Base Approach

In early computer systems, a typical approach was to set up one or more separate files for each data-processing application. Although this was somewhat satisfactory for each application, it led to several files with duplicate data, causing serious update problems. The separate-file approach also made it difficult to extract information that was scattered through various files.

To meet such information needs, the concept of a data base was developed. Under this approach a file is not treated as a separate entity and individual files are not set up for use by just one program. Instead, data needed by many different data processing applications is consolidated and integrated into a common 'pool'. Numerous definitions for the term database exist in contemporary literature. Some of them are stated here to further explain the concept of a data base:

1. A data base is a collection of stored operational data used by the application systems of some particular enterprise. [2]

2. A database may be defined as a collection of interrelated data stored together without harmful or unnecessary redundancy to serve multiple applications; the data are stored so that they are independent of programs which use the data; a common and controlled approach is used in adding new data and in modifying and retrieving existing data within the data base. The data is structured so as to provide a foundation for future application development. [3]

3. A database is a set of 'integrated files containing current data on company personnel and resources, the organizational environment, the competitive situation, and so on'. [4]

For the purpose of this project a definition of a data base is given in broad, general terms viz., A data base is an integrated source of data which is accessed by many users and is controlled by a Data Base Management System (DBMS). In turn, the DBMS is a package of software programs designed to operate interactively with a collection of computer-stored files or data base. The functions of the DBMS are:

(i) Data Base creation - defining and organizing the data needed to support an information system.

(ii) Data Base maintenance - adding, deleting, updating, correcting and protecting the data in the data base.

(iii) Data Base processing - utilizing the data in a database to support various data processing assignments such as information retrieval and report generation.

The data base management system is designed so as to allow the user to deal with the data in abstract terms (ie. logically) rather than as the computer stores the data (ie. physically).

A data base system is a system that includes both a data base and a data base management system. A summary of the state of the art in data base management is presented in a recent issue of Computing Surveys [5].

1.2 Advantages of Data Base Systems

Data Base Systems were developed to overcome a number of shortcomings in existing file management systems. Advantages of data base systems can be listed as follows:

1. Reduction of data redundancy - the proliferation of data unavoidable in traditional file management systems is eliminated, and redundant data are maintained only when it is required or economically beneficial. Independent files are eliminated, so storage requirements are reduced.

2. Increased capability to relate associated data - this allows the production of reports for multiple, interrelated files, a requirement difficult to meet in the traditional file-oriented approach.

3. Independence between application programs and data - changes in the form or representation of the data or in the relationships between the data do not affect the application programs. This kind of data independence is referred to as physical data independence.

4. Increased data integrity - certain kinds of consistency constraints (ie. required properties of the data) can be checked by the DBMS if it is told to do so. The application programmer has no direct way to change physical data bases, so it is almost impossible to destroy data, alter relationships between data, add duplicate records, lose records, etc.

5. Crash protection and recovery - to protect the data base against accidental loss, facilities to make regular backup copies of the data base and to reconstruct the data base after a hardware or software error are provided.

6. Security of data - access to data can be easily limited by passwords to authorized users, so data privacy can be ensured.

7. Adaptability and flexibility - physical data can be viewed as logically different by different users. This permits multiple uses of the same data.

8. Interface between user and the data base through a high level, non-procedural language. This language is simply a notational language for interrogating the data base. It must allow the casual user to deal directly and effectively with the data base, without consulting a programmer.

1.3 The Three Data Models

Before one can attempt to understand, design, implement or even use a data base management system, one needs a data model. A data model is a representation of the entire information content of the data base, in a form that is somewhat abstract in comparison with which the data is physically stored. Many data models have been proposed, each with its own concepts and terminology.

The three principal data models used in data base systems are:

1. The Network Data Model - a major commercial system based on this model is TOTAL [6]. Others using this model are Honeywell's Integrated Store (IDS), ADABAS (Software Ag), Univac's DMS-1100 and Cullinane's IDMS.

2. The Hierarchical Data Model - IBM's Information Management System (IMS) is one of the most heavily used of commercially available systems and is partially responsible for the importance of the hierarchical data model on which it is based [7]. Another example of a DBMS using this data model is System 2000.

3. The Relational Data Model - This is exemplified in the experimental System R data base management system [8], RDMS and RISS.

A suitable reference for data base system vendors is given in the Bibliography [9].

The Hierarchical and Network data models are the most widely used in present data base systems. The Network model is characterized by the Codasyl Data Base Task Group (DBTG) proposals [10]. This data model provides a relatively high performance but requires the user to specify storage structures, access paths and data structures in considerable detail. The network model is also characterized by inflexibility, in that access paths not predefined at data base load time can never be used. The storage structures are generally constructed of pointers to linked lists and tend to be quite complex.

The Hierarchical approach, like the network approach is based on a tree structure. But whereas in the network model a child node can have more than one parent nodes, in the hierarchical model a child node is

restricted to only one parent node. As with the network data model, the hierarchical model requires understanding the use of pointers and linked lists and is difficult to modify once it is set up. The hierarchical structure works well with some data bases but it becomes difficult to design data bases using a hierarchical data base system when a natural hierarchy among record types does not exist.

In contrast, the Relational approach generally shields the user from the complexity of storage structures, data structures and access paths. Access paths need not be predefined . All data within the data base is viewed as being in simple tables. Each table is a relational model of actual data relationships; at the same time, it is a structure that can be easily understood and one that is suitable for display on visual-display units. The data base system that supports this approach performs well defined mathematical operations upon the data base as normalized relations [11], but the details of these operations need not be coded by the application programmer.

Although there are few implementations of the relational approach at present, it is expected to be the primary model in the future. Hence, in choosing a data base management system for this project, one based on a relational model of data was selected.

1.4 Choice of DBMS

The Relational data base approach has only recently begun to attract attention. It is being considered, debated and investigated for future Data Base Management Systems (DBMS). Some of the most recent papers in this field are those of E. F. Codd [12], Whitney [13], Chamberlain [14] and Astrahan [15].

At present, most of the available DBMS are those based on the hierarchic and network data models. As a result choice of a relational DBMS for possible conversion and use on a microcomputer was extremely restricted. The RISS (Relational Inquiry and Storage) data base management system was selected because of its availability and because it was designed for a minicomputer as opposed to a large main frame.

The RISS software consists of an Editor, a Retrieval Package and a data base Maintenance and Manipulation Package to suit the needs of the casual user. For the application programmer, RISS provides a set of primitive functions that enable access to and modification of the data base.

RISS used the relational approach as opposed to the network and hierarchical approaches because it provided the following advantages:

1. It provides a simpler, more unified, user data model, resulting in systems that are easier to use and maintain.

2. It is much more data-independent and consequently results in systems that are more generalized. In addition, relational data bases are easier to alter, eg., when new data relationships are discovered.

3. It is much easier to express data semantic integrity constraints (limitations on the permissible data in a data base).

4. Data retrieval and modification requests are easier to express (in a generalized manner). These requests may be expressed in a way that is much less procedural.

5. Since information is presented in one and only one way in a relational data base, only one operator is needed for each of the basic functions (store, retrieve, etc.) to be performed.

6. The emphasis is on the use of sets (in the mathematical sense, not the Codasyl Data Base Task Group sense), rather than on handling one record at a time.

7. Sharing and protection requirements are more easily satisfied, due primarily to the simplicity of the underlying data base model and absence of highly distributed access paths.

8. Implementation issues are isolated from the logical data base model. This results in increased intersystem compatibility and, most

significantly, encourages a structured approach to implementation.

1.5 The Relational Data Model

The Relational model is a mathematical approach based on the set theoretic notion of a relation. The logical structure used is the Relationship in Third Normal Form, which is the type of relation with the optimal properties for use in the database [11].

All data in the relational model is viewed logically as a simple table. This is easily understood by the layman, and is suitable for display on terminals. Mathematically these tables are known as relations or, strictly speaking, relationships. A relation of degree 'n' has the following properties:

1. It contains 'n' columns (known as domains).
2. All elements in a given domain are of the same type.
3. The ordering of columns is of no significance, since all columns are labelled.
4. Each row represents an n-tuple of the relation and contains 'n' elements.
5. The ordering of rows is immaterial.
6. All rows are distinct (there are no duplicate tuples).
7. Columns are assigned distinct names.

In conventional terms, a relation can best be equated to a serial file containing one record type of fixed length. Thus, a tuple is equivalent to a record; a domain, to all data items of a particular type in the file. Tuples are identified by their keys, which are formed from a combination of one or more data-items. The primary key is defined as that data item or combination of data items used to uniquely identify one tuple. The primary key is of great importance because it is used by the computer in locating the tuple by means of an index or other addressing technique.

The totality of data in a relational data base may be viewed as a collection of time-varying relations. These relations are of assorted degrees. As time progresses, each n-ary relation (of degree n) may be subject to insertion of additional n-tuples, deletion of existing ones, and alteration of components of any of its existing n-tuples.

The relational model permits a concise definition of the contents of the data base. This description is called the schema. Since the user is not concerned with ordering, indexing or access paths, the schema is defined by naming only the relations and the domains of the relations and indicating their primary keys.

1.6 Choice of Language

The programming language BASIC was chosen to implement the microcomputer DBMS for the following reasons:

(i) it is a business-oriented language,

(ii) it is an easy to learn, easy to use, interactive programming language,

(iii) it was available on the Dynabyte microcomputer,

(iv) to avoid making this project unmanageable, it was necessary to stay as close as possible to the language used in the original software.

The programming language PASCAL was a likely choice since it is available on most microsystems. It allows for more structured programming than BASIC and has a rich domain of data structuring facilities. However its limited Input/Output capabilities made it a less likely candidate than BASIC.

Two versions of BASIC were available on the Dynabyte microcomputer viz., MICROSOFT BASIC and CBASIC-2. MICROSOFT BASIC was designed specifically for use on microcomputers and includes features that enable the user to access the data in any memory location and read it (PEEK) or substitute something else (POKE). However, CBASIC-2 was selected above MICROSOFT BASIC for the following reasons:

(i) Microsoft Basic lacked the facility for defining multiple-line functions, a facility which was vital to the project, since the RISS application-level interface (see Chapter II) consists of a set of multiple-line functions.

(ii) Random access files were used to store the data in a RISS relational data base because a record can be located much more quickly, without having to read through all the information on the disk. In MICROSOFT BASIC, random access files are stored in packed binary format, thus requiring less room on the disk. However, creating and accessing random files in MICROSOFT BASIC is much more complex than in CBASIC-2. To create a random file in MICROSOFT BASIC, the following program steps are necessary:

. Open the file for random access ('R' mode)

. Use the FIELD statement to allocate space in the random buffer for the variables that will be written to the random file.

. Use LSET to move the data into the random buffer. The data must first be converted to strings using the 'convert' functions provided.

. Write the data from the buffer to the disk using the PUT statement.

In CBASIC-2 buffer space for files is allocated dynamically. Only two

program steps are therefore necessary.

- Open the file with a fixed record length indicating that access is random.
- Write data to the file using the PRINT Statement.

3. MICROSOFT BASIC lacked some of the string manipulation functions that were available in BASIC-PLUS, the version of BASIC used to implement the original data base management system.

1.7 A Microcomputer Data Base Management System

The development of microcomputers represents a major revolution in computer science and technology due to accelerating trends in micro-electronics. The microcomputer is a very small computer, ranging in size from a 'computer on a chip' to a small typewriter size unit. Thus, computers of extremely small size and cost but yet of great speed, capacity and reliability are now a reality.

There are many organizations that have application environments requiring data base management capabilities but cannot afford spending large sums of money on a computer system. Microcomputers have become increasingly important in applications that cannot afford the cost and do not require the capabilities of a larger minicomputer. However, because microcomputers are more recent than minicomputers, their

software library is smaller than that of minicomputers. The development of a microcomputer data base management system, therefore, is a worthwhile contribution towards filling the gap that exists.

CHAPTER II

STRUCTURE OF RISS DBMS FOR MICROCOMPUTERS

A microcomputer data base management system has been developed for creating, accessing and maintaining logically related files which make up a data base. This chapter describes the structure of the converted RISS data base management system. However, for the sake of simplicity, the converted DBMS will also be referred to as RISS.

2.1 Logical Structure

Each RISS database consists of a set of normalized relations. A normalized relation may be viewed by the user as a two-dimensional table, where each row of the table corresponds to a tuple (record) of the relation, and each column of the table has a unique name and an underlying domain. This domain is the abstract set of data values from which entries in that column may be selected. In RISS, the underlying domain of a column is not explicitly specified. The user is only permitted to specify whether the data stored in a column is integer or character string.

Each data base relation is created by naming the relation and its constituent columns, and specifying the underlying domain of each column. More than one column in a relation may have the same underlying domain. In RISS, the concept of primary key (ie., a set of columns that uniquely identifies tuples in the relation) is not strictly necessary. Normally, relations are ordered in some way. In this implementation, no more than three relations can be accessed by a program simultaneously. A further restriction is that each tuple in a relation may have no more than 25 columns.

2.2 Physical Structure

The data base system is based on the representation of data as relations, which is the only structure provided. Both the data and relationships among the data are represented as relations. Each RISS relation is stored in three disk files: a Tuple Descriptor File (TDF), a Tuple File (TF) and a Column Descriptor File (CDF).

The Tuple Descriptor File (TDF) is a CBASIC-2 random - access Record I/O file. This file stores three pieces of information that are very vital to programs that perform operations on the data base. The three pieces of information are stored in one record, thus the tuple descriptor file is a file consisting of one record. The first two

fields of this record store the number of tuples (rows) and the number of columns (domains) of the relation, respectively. This information is used in processing the data base.

The third field of this record stores the maximum length of a tuple (or record) and is used to optimize storage utilization. In defining a relation, the user is asked to specify the name of each column, a strategy of 1 or 2 indicating whether the data is numeric or alphanumeric, and also the maximum field length for each column. The maximum number of columns permitted in a relation is 25. In the Tuple file all data is stored as strings within quotation marks and separated by commas. The maximum record length (RL%) is therefore calculated as:

RL% = (25 * 3) + sum of individual field lengths + 50

This number represents the maximum number of bytes needed to store the tuple. Fifty extra bytes are included to allow for adding columns to or deleting columns from a relation, as provided for by the COLREL utility (described in a later section in the current chapter).

The Tuple File (TF) is a CBASIC Random-access Record I/O file. The length of the record varies for each relation and is calculated when a relation is created as explained above. The maximum number of fields allowed in a record is 25. The tuple file contains the actual

data stored in a relation. This file is modified when insertions and/or deletions are made and is used for printing reports.

Each record in the tuple file consists of a number of fields delimited by quotes and commas. Adding a tuple to the tuple file consists of first printing a record consisting of 25 fields with null data, and then replacing the appropriate null strings with real data. Building the data base is therefore a time consuming process.

The Column Descriptor File (CDF) is a CBASIC Random-access Record I/O file. It stores information about the columns of a relation. Each record in the file contains three pieces of information viz., the name of a column, the column strategy, the field length for that particular column. The name of a column can be any alphanumeric character string of arbitrary length. The column strategy is used to distinguish the type of data found in the different columns. A code number of 1 is used to indicate the presence of only numeric data in a column. A code number of 2 indicates the presence of only alphanumeric character strings in the column. This information is important in performing calculations on data in the data base. The value of numeric data must be obtained before any calculations can be done on the data. The field length represents the maximum number of bytes necessary to store any data item in a particular column. One byte is allowed for each

character stored.

Random-access file organization is used in all three files described above, because it provides a fast and easy way to access any record in a file. In random access the program is not limited to accessing the next record or field, as in sequential access. Any record in the file is as accessible as any other. Each record, or position where a record may be placed, is referenced by its relative record number.

2.3 Data Manipulation

The RISS data base management system provides a set of common functions which are used to process the data base through a common user interface. Typically, the set of functions includes definition, creation, interrogation and update. The interface between the system and the user has a dual purpose

- to permit the user to define the relationships among data items; and
- to provide the user with a facility for using meaningful subsets of the database.

RISS is a host-language system, the user interface being a collection of data manipulation procedures written in the host language (CBASIC-2). The RISS routines have been developed for two different classes of

users: the naive user and the application programmer.

1. Naive users are routine users of the system, such as data clerks, managers etc., who are not interested in learning a programming language. Interface at this level are based on a set of commands that enable the naive user to access and modify the contents of a data base relation, without having to know any unnecessary details about the data base.
2. Application programmers require the ability to access and modify a data base. The RISS DBMS provides a set of primitive functions to interface between the programming user and RISS relations and between the RISS data structures and the CBASIC Record I/O structure.

These two interface levels are discussed further in the following sections.

2.3.1 The Naive-User Interface Level

The facilities provided by the DBMS for the naive user consist of the following:

1. a relation editor
2. a retrieval package
3. a data base manipulation and maintenance package

1. The Editor

The relation editor (program name is EDREL) allows the non-programming user to enter, examine or modify data in a RISS relation. The editor also allows a user to add and/or delete tuples (rows) of a relation. The user interacts with this module using a set of built-in commands.

The editor is based on that of a line-oriented text editor. A relation is viewed as an ordered list of tuples. A 'current tuple pointer' is maintained, which points to the first tuple in the relation when the editor is entered, and which may be moved by the editor commands so that it points to the desired tuple in the relation. The editor contains commands that allow the user to:

(i) move the pointer an integral number of tuples forward or backward

(ii) move the pointer by searching a column for a specified value (eg., for character strings via an 'exact match' or 'substring' search)

(iii) delete one or more tuples after (and including) the current tuple

(iv) insert a new tuple after the current tuple

(v) display or change the value of a column of the current tuple

(vi) provide descriptive information about the relation (eg., the name of a specific column).

After the EDREL command module has been accessed, the user is asked to specify the name of the relation to be edited. Once this is done, two display parameters are requested by the editor. First, the user has to indicate if the column name is to be included whenever a data item is displayed. If so (ie. the user types 'Y'), then the corresponding column number and name are shown along with any data item. If not (ie, the user types 'N'), only the column number appears with the data. For this parameter the default response is 'YES'. The second parameter is the number of data columns to display whenever the current tuple pointer is moved. The user indicates the number of columns to display and the desired column numbers. For this parameter, the default response causes column 1 to be displayed when the pointer is moved. Once these parameters are entered, the editor sets the tuple pointer to the first tuple of the relation, and prints an '*' to indicate that it is in the command mode.

In the command mode the user may select any of the following 13 options:

? (Status) : Give the location of the current record pointer.

C (Columns) : List the name of each column in the relation.

+ (Plus) : Move the current tuple pointer forward a specified number of rows. The user may specify this number in the command line (eg., '+5') or, if only + is entered, the editor will query for the number of tuples to advance the pointer. If the number indicated would place the pointer beyond the length of the relation, the current tuple pointer is set to the last tuple in the relation. This is useful for getting to the end of a relation.

- (Minus) : Move the tuple pointer backward a specified number of rows. As with the Plus option, this number may be included in the command line (eg., '-9'). Attempting to move the pointer beyond the range of the relation causes it to be set to the first tuple of the relation. This is useful for getting back to the beginning of a relation.

L (Locate) : The Locate option is used to find the first occurrence of a character string in a specified column. The user enters the column to search and the string to be located. The search begins at the

current tuple and continues to the end of the relation; it continues, if necessary, at the first tuple of the relation until the current tuple is reached. If the indicated string is not found, a report is printed to that effect and the current tuple pointer remains unchanged. If it is found, the pointer is set to the tuple containing the indicated string in the column searched.

S (Substitute) : For the Substitute option, the user enters the column to search and a character string to locate (search string). Then as in the LOCATE option, the editor moves the tuple pointer to the tuple where the column data contains the search string. The user is then asked for a substitution string. Once this is supplied, each occurrence of the search string is replaced by the substitution string in the located data item.

E (Examine) : The Examine option enables the user to examine the contents of a column in the current tuple. After the column number is specified, the contents are displayed.

V (Value) : With the Value command, a user may modify the contents of any column in the current tuple. Once a column is selected, the current contents are displayed and the user is prompted to enter the new data. The new data is then stored in that column location of the current tuple.

I (Insert) : The Insert option is used to add tuples to a relation. A tuple may be inserted between two existing tuples or added at the end of the relation, using this option. A record with null data is first added after the current record. When this is done, the number and name of each column are displayed, and the user may enter the data to be stored in the new tuple.

B (Bottom) : The Bottom option functions as the Insert option, except that the new tuple is always added at the end of the relation.

P (Print) : The Print option prints a specified number of tuples of the relation, beginning at the current tuple and continuing until the specified number of tuples has

been printed or until the end of the relation is reached. The current record pointer is reset to the last tuple printed.

D (Delete) : The Delete command allows a user to delete one or more tuples from a relation. The number of tuples may be entered as part of the command line (eg. 'D5'). The delete operation begins with the current tuple and continues until the specified number of tuples has been removed or the end of the relation is reached. In this implementation, there are no safeguards against accidental deletions. The user is responsible for ensuring that the right tuples are being deleted, possibly by examining the records before deletion. After a deletion, the current tuple pointer points to the tuple immediately following the deleted tuple (or tuples). If the last tuple in the relation is deleted, then the tuple pointer points to the tuple immediately preceding the current tuple.

With the delete command, tuples are not deleted immediately, but a tuple number pointer in array M2

is modified. The Array M2 is an integer array that is used to store the number of each tuple (row) in the relation. Initially each array element contains a corresponding tuple number. If an insertion or deletion is made, the tuple number and the number position of the array element no longer correspond. This indicates that the tuple file must be rebuilt before terminating the editing process. Because of this, deletion must be the last operation performed on a relation, before returning to the operating system.

Q (Quit) : Terminate editing and return control to the CP/M monitor. If any tuples were inserted in or deleted from a relation, the tuple file is first rebuilt.

2. The Retrieval Package

The RISS retrieval package (program name is RETREL) allows the user to retrieve and analyze the data in RISS relations. It provides commands that facilitate:

(i) selection of a set of retrieved tuples (retrieved set) on the basis of a column value comparator (eg., sex = 'male', Age > 18),

(ii) modification of the retrieved set by forming the union or intersection of the retrieved set with a new set of tuples selected by a column value comparator,

(iii) extraction of a subset of the columns of the tuples of the retrieved set,

(iv) printing a tabular report based on the retrieved set,

(v) printing simple statistical information (eg., mean, median for numerical data) for the retrieved set,

(vi) forming several groups of the data in a particular column by common value (eg., STUDENT tuples grouped by SECTION), or by specifying a range of values for each group (eg., marks 0-45; 45-60; etc.) and obtaining information about the tuples in each group (eg., a list of all of them or the number of tuples in each group),

(vii) producing a list of all the distinct values of a particular column for the retrieved set and the frequency of occurrence of each distinct value.

Complex combinations of these operations are also possible.

The command module is initialized after the user enters the name of the relation from which data are to be retrieved and specifies whether reports should be output to the printer or displayed on the

screen. If the reply is 'P', reports are sent to the printer. Once initialized, RETREL prints an '*' on the user terminal to signify that it is in command mode.

In command mode there are 10 user options:

Q (Quit) : Terminate the retrieval operations and return to the CP/M monitor.

C (Columns) : Print the name of each column in the relation.

A (And) : The And command is used to select subsets of the relation's tuples (records). With this command the user is asked to supply the criterion for completing the tuple-selection process. All criteria are based on the data contained in a given column of the relation, or on the row number of a tuple in the relation. The tuple selection criterion allowed is that the data in a column, or a row number, must be less than, equal to, or greater than a specified comparator item. The user is prompted by the command module to supply the column number for the criterion (column zero is used to signify the tuple number); the comparison mode (less than, equal to, or greater

than); and the data item comparator. If the comparison mode selected is 'equal to', the user is also queried as to whether or not an exact match is required. If so, then the column data must be exactly equal to that comparator item. If not, then the criterion is satisfied if the comparator item is a substring of the column datum.

For data columns containing numeric data and for row-number criteria, numerical comparisons are performed. For columns with alphanumeric data, character string (lexicographic) comparisons are used.

When RETREL is initialized, all tuples in the relation are flagged as active (retrieved). With the AND command the user indicates that he or she wishes to consider active only those tuples that were active at the time the command was issued and that also satisfied the entered column criterion.

O (Or) : This command functions in the same manner as the AND command. The user specifies a selection criterion and those inactive tuples that satisfy the criterion

are added to the list of already active tuples.

I (Initialize): The I command initializes the relation and converts all tuples to the 'active' state.

S (Statistics): When the S command is given, the user is asked to enter a column number. The module then prints out the number of currently active tuples along with the mean, median, and (if desired) the standard deviation for the data in the indicated column.

P (Print) : The Print command allows the user to print out in tabular for one or more columns of each currently active tuple. The user enters the number of columns to print and then the column numbers to be printed. Indicating column zero causes the relation tuple numbers to be printed. The relation retriever will than ask if the user would like the data to be printed in aligned columns. If so tab positions (not less than one) for each column must be entered.

T (Tally) : The Tally command allows the user to compile a table of all data items stored in a relation column for the currently active tuples. The table includes each of

the different data items present, the total number of occurrences for each data item, and the percentage of the number of active tuples which that frequency represents. The number of tuples with null data in the specified column is also printed.

G (Group) : This command enables printing of a table of frequency counts for groups based on column data. The user specifies the data column, number of groups and the upper bound for each group. Again, only currently active records are considered.

M (Move) : With this command, all currently active records are moved to a user-designated relation. The user may then indicate whether or not the moved records are to be deleted from the original relation.

3. The Maintenance and Manipulation Package

The RISS data base maintenance and manipulation package allows the user to perform various operations necessary for the maintenance and routine use of a data base. All of these operations need not be accessible to all users; these operations are probably most useful to the data base administrator (ie., the authority responsible for maintaining the data base).

Facilities are provided for:

(i) creating a relation

(ii) deleting a relation

(iii) copying a relation

(iv) sorting a relation

(v) merging two relations

(vi) combining two relations via a 'join'

(vii) adding a column to an existing relation

(viii) deleting a column from an existing relation

These are discussed in detail in the following sections.

CREREL - Creating Relations

This module enables the creation of a RISS relation. After accessing this module, the user is asked to supply the name of the relation to be created. Specification of the physical storage location of the CBASIC files that comprise the relation is then possible. The user then indicates the number of columns in the relation.

At this point the user is permitted to define the name, strategy and field length of each column. The name of a column may be any alphanumeric character string of arbitrary length. The strategy is a code number which is used to indicate the type of data stored in a

particular column. A strategy of 1 indicates that the column will contain only numeric data; a strategy of 2 means that the column will contain alphanumeric information. After all columns are defined, the information entered may be edited.

When this is done, a report is printed to the effect that the specific relation was created.

COLREL - Maintaining Relation Columns

The Colrel module allows a user to perform three distinct column maintenance functions:

- alter names of columns
- delete columns
- add columns to a relation.

After accessing the command module and entering the name of the relation to be maintained, the user selects one of the above three functions.

In changing a column name, the user has to specify the column number and the new name for the column. For deleting a column, the number of the column is entered. To add a new column, the user enters the number of the column after which the new column should be added as well as the name, strategy and field length for the new column. A field with null data is inserted in all of the tuples in the correct

position.

MERREL - Merging Relations

The relation Merger allows a user to create a new relation (join) from the contents of two existing relations.

After accessing the Merrel command module, the user is asked to enter the names of the two relations to be merged, as well as the name of the relation to be created by the merge. The user is then queried as to the number of columns in which the data is the same for the two relations (the number of comparison column pairs). Following this, the user is required to specify the numbers of these columns that match in the two relations. A tuple in the merge relation will then be created for each pair of tuples in the source relations, that match exactly in all comparison pairs. Merrel will then create a relation of order N (ie., N columns) where

N = order of the first relation + order of the second relation - number of column comparison pairs.

SORREL - Sorting Relations

The Sorrel module allows the user to sort the tuples of a relation based on the contents of some specified column in the relation. To sort relations on a column, the user accesses the command module SORREL, and specifies the column to sort on. This module sorts tuples

in ascending order on the specified column. For columns with numeric data, ie., strategy is 1, the sort is done on a numerical comparison basis; otherwise, sorting is done on alphanumeric character string (lexicographic) comparisons. The sort process writes the sorted records to another CBASIC Random-access Record I/O file. This sorted file then becomes the tuple file and the old tuple file is deleted. At the end of the sort, the user is notified that the sort process was completed.

COPREL - Copying relations

This module allows a user to make an exact copy of an existing relation. The relation copier is used in creating backup copies of relations.

When the module is accessed, the name of the existing relation and the name of the relation to copy to, must be specified. Care should be exercised in naming the copy relation since any existing relation with the same name will be overwritten. For each relation, the three files associated with it are copied to the new relation.

DELREL - Deleting Relations

With the relation-deleter module, a user can remove a relation from permanent storage. After the user supplies the relation name and confirms his request to delete that relation, the three disk files

associated with the relation are then deleted. A deletion report is then generated.

Accessing RISS Command Modules

The preceding command modules are accessed by the user by entering a command of the form:

CRUN2 MODULE

where MODULE is the name of one of the preceding 8 command modules.

eg., CRUN2 EDREL

Some samples of interactive dialogue demonstrating the use of these modules appear in Appendix A.

Relation Name Specification

Each command module requests the user to specify the name of one or more relations, at some point in the execution of the module. In naming relations, the same conventions used in naming files are adhered to, except that the file type is not specified. Thus a relation name consists of one to eight letters and/or digits eg., STUDENT, CLASS. If the relation does not reside on the currently logged drive, then the user must specify the drive (A or B) containing the diskette on which the relation resides. The drive letter is separated from the rest of the name with a COLON.

eg., B:STUDENT, A:CLASS

2.3.2 The Application Program Interface Level

The application program interface consists of a set of callable operations or RISS functions. These functions may be used by any application program, and may return information to the calling routine or make changes to a RISS data base relation, or both.

The RISS functions are callable from programs written in the language (host language) used to implement RISS, the same language that is used to write RISS applications programs. In this project, the host language is CBASIC-2. In effect these functions define an extension of the host language. Functions are provided to facilitate:

1. the initialization of relations
2. the addition and deletion of tuples of a relation
3. the return of information about the columns of a relation
4. the return or alteration of the value of a specified column for a particular tuple in the relation.

The RISS functions operate on a low level, and, unlike the naive user interface level, their basic structure is implementation - dependent. This is because they actually manipulate the relation representation.

Appending RISS Function Code to Programs

Each RISS function is stored on disk as a file of type BAS. eg., FNR1.BAS. To insert the RISS functions into the source code, the CBASIC-2 %INCLUDE feature must be used.

e.g., %INCLUDE FNR1

This causes the compiler to compile the file, specified in the include statement, into the source immediately following the %INCLUDE directive.

RISS Functions

To interface between the programming user and RISS relations and between the RISS data structures and the CBASIC Record I/O structure, RISS provides 13 functions. Seven primary functions enable interaction between the programmer and RISS relations, and six secondary functions interface between RISS and CBASIC.

Secondary Functions

FNR5% - reads data from the relation's Tuple Descriptor File (TDF). The function has one parameter, which is used to select the file from which data is to be read. The data in the three fields of this single-record file are assigned to the variables NT%, NC% and NL% for use in any application program. NT%, NC% and NL% store the number of tuples, the number of columns in a

relation and the length of a relation tuple, respectively. The function returns a value of zero.

FNR7% - stores data in a relation's TDF. This function has one parameter which is used to select the file to which data is to be written. The function stores the number of tuples (NT%), the number of columns (NC%), and the length of a relation tuple (NL%) in the single record of the TDF. To alter a field in this record, it is necessary to read the whole record using function FNR5%, assign the new value to the appropriate field, and then rewrite the whole record using function FNR7%.

FNR9$ - is used to read a record from any CBASIC Record I/O file. All fields of a record are read into the array T$. The parameter I% represents the record to be read and the parameter F% is the number associated with the file from which data is to be read.

FNR0$ - is used to write a record to any CBASIC Record I/O file. The parameters correspond to those used in FNR9$.

FNR5$ - is used to store an entire tuple in the relation's Tuple File. The function has two parameters: I% represents the tuple to be stored and R% the logical number of the relation. This is

used to calculate the file number associated with the Tuple File.

FNR7$ - is used to read an entire tuple from a relation's Tuple File. the parameters correspond to those used in FNR5$.

Primary Functions:

FNR1$ - This is a two parameter function, used for the purpose of initializing relations. The first parameter is the name of the relation to be initialized; the second parameter is the logical number 1, 2 or 3 of the relation in the program. Relation 1 is assigned file identification numbers 10,11 and 12. Relation 2 is assigned numbers 7 through 9 and Relation 3 is identified by file numbers 4 through 6.

R7% (R%) is set to one less than the file number of the Tuple Descriptor File for logical relation I%. The three files of the relation are then opened with the appropriate file identification numbers. Information from the relation's Tuple Descriptor File are then made core resident, so that:

R2% (R%) = length of the relation (number of tuples)

R3% (R%) = order of the relation (number of columns)

R4% (R%) = length of a relation tuple in bytes

R4% (R%) = length of a relation tuple in bytes

For each relation opened R% = 1, 2 or 3.

FNR2$ - This primary function has three parameters I11%, I22% and R% and is used to retrieve the data item stored in Row I11% and column I22% of the relation with logical number R%. R% is used to calculate the logical file number; this number pertains to the Tuple File (TF).

FNR2$ uses secondary function FNR7$ to read the whole tuple into the buffer. The required data item ie., T$(I22%) is returned by assignment to the function name FNR2$. All data are returned from a relation in character string format. If the value of data is desired for use in calculations, the CBASIC-2 function 'VAL' may be used to obtain the desired conversion.

FNR3$ - Function FNR3$ is used to store data in a particular column of the relation tuple file. This function has four parameters: I11%, I22% and R% which correspond to the parameters in function FNR2$; S$ is the data string to be stored.

Secondary function FNR7$ is used to read the desired tuple from the relation. S$ is stored in the required field and the whole tuple is written back to the file using secondary function FNR5$.

FNR4$ - Function FNR4$ is used to read the relation's CDF (Column Descriptor File) and return information about the column C% in the relation with logical number R%. The function reads the data in record C% into the variables N$, S$, F$. These variables are used directly in any application program. They store the name, strategy and field length for the column C%.

FNS4$ - Function FNS4$ is used to store data in a relation's Column Descriptor File (CDF). This function has two parameters; the column number C%, and the logical number R% of the relation. The purpose of the function is to write the values of N$, S$, F$ to the record C% of the CDF. To alter a field of this record, it is necessary to use function FNR4$ to make the values of the fields available to the program; assign the new value to the appropriate field - N$, S$ or F$ and then rewrite the record to the CDF.

FNR6$ - Primary function FNR6$ adds a tuple to relation R% after an existing tuple I%. It first determines whether or not the tuple is to be added to the end of the relation or inserted at some intermediate point. If the new tuple is not the last one in the relation, the function transfers the contents of each

tuple one position forward from the insertion point onward. A tuple with null data is then stored in the new location. The relation length parameter in the Tuple Descriptor File is then reset.

FNR8$ - Function FNR8$ deletes tuple T% from relation R%. This is achieved by transferring the data in tuples following the deleted tuple backward one position. The relation length parameter in the Tuple Descriptor File is then reset.

2.4 Integrity and Privacy Controls

The integrity of the data base is a user responsibility. The system provides no procedures for validation of input data. No recovery procedures or privacy locks are provided. Regular backup copies of relations can be made using the COPREL utility.

CHAPTER III

DIFFERENCES BETWEEN THE PUBLISHED VERSION AND THE MICROCOMPUTER CBASIC-2 VERSION OF THE RISS DATA BASE SYSTEM

The software that constitutes the RISS data base management system was originally developed and tested in BASIC-PLUS on a DEC PDP-11 minicomputer, which ran under the control of the RSTS/E operating system. In this project, the DBMS software was converted, for microcomputer use, to the CBASIC-2 version of the software that appears at the end of this report.

Before making any changes to the software, it was necessary to become thoroughly familiar with the published version. This meant knowing generally what each program did and how all the programs worked together. It also meant knowing the detailed working of each program that was to be changed. It was necessary to know what was on each data file and how the data files interacted. Several changes were made to the software resulting in an end product that is significantly different from the original version of RISS. These differences are discussed in the following sections.

3.1 Physical Data Organization

In the published version of RISS, each RISS relation is stored in three on-line disk files: a tuple descriptor table (TDT), a tuple file (TF), and an alpha data file (ADF).

The TDT contains two virtual integer arrays: a one dimensional three-element array, and a 32767 by five-element two dimensional array. The tuple descriptor table contains pointers and keys used in storing and retrieving data from the tuple file and alpha data file.

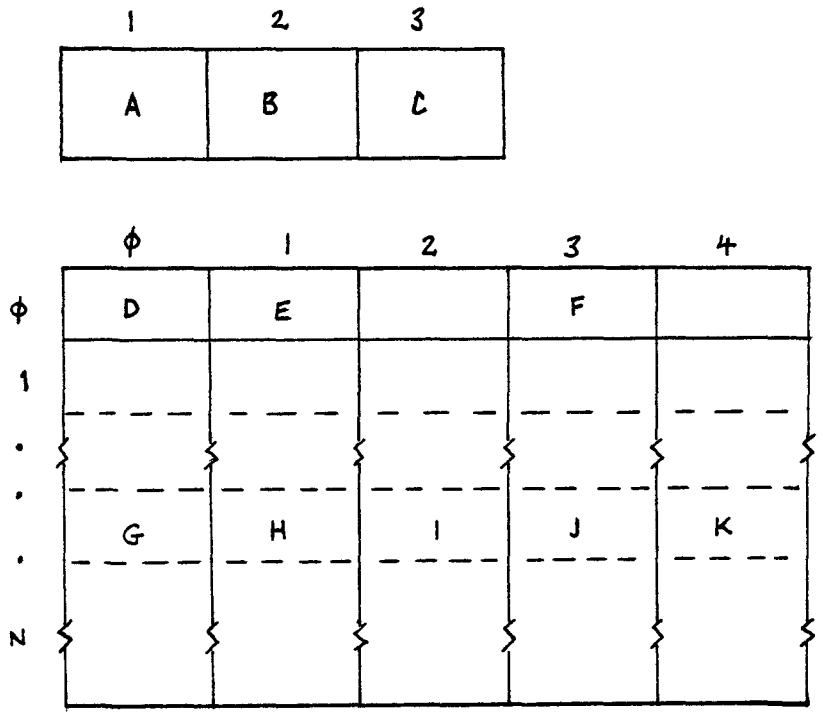


Figure 3.1 Tuple Descriptor Table Arrays

The contents of the tuple descriptor table shown in figure 3.1 are:

Location A: Record (block) number of the next free byte in the ADF.

Location B: Offset in that block to the byte.

Location C: Length of a relation tuple in bytes.

Location D: Number of tuples or rows in the relation (ie., the relation length.

Location E: Order of the relation (ie., number of columns in the relation).

Location F: These three integers are used to code the physical distribution of the three Basic-Plus files in the disk-storage system.

The remaining rows in the two dimensional array hold parameters for the relation columns. Each row in the array corresponds to a column of the relation.

Location G: Relation-column strategy.

Location H: Offset to the start of the relation - column storage, in bytes, within a tuple eg., for the sample tuple in figure 3.2 the four values stored in Location H would be 0, 1, 3, and 7).

Location I: Block number pointer to the storage location of the column name in the alpha data file.

Location J: Offset for the column name.

Location K: Length of the column name in bytes.

The ADF (alpha data file) is used to store variable-length alphanumeric character string data. The location of data strings stored in this file is given by the logical record number of the block containing the start of the data string, the offset in that block to the start of the string, and the data-string length.

The TF (tuple file) is composed of fixed-length N-tuples, which correspond to rows of the relation. There is a one-to-one correspondence between N-tuples and rows in the relation. The length of an N-tuple (in bytes) is the sum of the data-storage formats (data strategies) defined for the relation's columns. RISS allows four data-storage strategies. Strategy 1 is used for storing single ASCII characters. Strategy 2 allows storage of integer numbers in the range -32768 to +32767. Data columns with a strategy of 4 are used for floating point numbers, and strategy 6 columns hold variable-length alphanumeric character strings.

For example, a sample relation with four columns having data strategies 1, 2, 4 and 6 is composed of 4-tuples each with a length of

13 bytes (as in Figure 3.2).

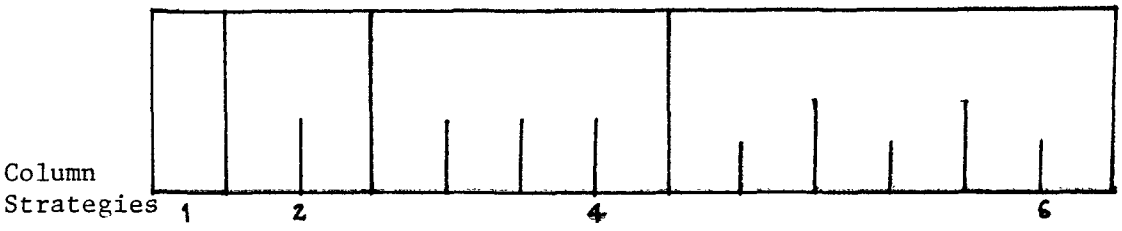


Figure 3.2 A sample 4-tuple

For strategy 1 columns, the single ASCII character is stored in the tuple byte associated with the data column. For strategies 2 and 4 data are converted, using the BASIC-PLUS CVT%$ and CVTF$ functions, into two- and four-byte length character stings, respectively and then stored in the corresponding tuple location. Data in columns with a strategy of 6 are stored in the alpha data file, and the three integers indicating its location (ie., the block number containing the starting byte of the string, the offset in that block, and the string length) are converted to successive two-byte strings by the CVT%$ function and then stored in the allocated six-tuple bytes.

3.1.1 RISS Storage Structure - CBASIC-2 Version

After a very careful study of the information stored in the arrays and data files used in implementing the original RISS data base

management system, it was decided that, for the microcomputer version, three files would also be used for storing each relation viz., (i) a tuple descriptor file (TDF) (ii) a tuple file (TF) (iii) a column descriptor file (CDF). Unlike BASIC-PLUS, CBASIC-2 does not provide virtual array storage facilities, and therefore all three files were CBASIC-2 Random-access Record I/O files.

As previously explained in chapter 2, The Tuple Descriptor File (TDF) is a random access file, used for storing vital information for processing the data base. The three fields of information are stored in one record and include the number of tuples in a relation, the number of columns, and the calculated length of a relation tuple (in bytes). Data in CBASIC-2 random access files are stored as a series of ASCII characters, where one byte (a code of 8 bits) is necessary to store each character. In creating a relation, the user (normally the data base administrator) has to specify the maximum number of characters for data (field length) in each column. These field lengths are summed to obtain the maximum number of bytes needed for a tuple in the relation. These three pieces of information must be made core resident at the beginning of each data base application. They are read in automatically into the arrays R2%, R3% and R4% by the RISS function FNR1$.

The Tuple File (TF) stores the actual data in a relation. This file is a CBASIC-2 Random-access Record I/O file. All data written to this file is in ASCII character format. The contents of both string and numeric variables are written as their representative ASCII characters, not as binary data. All fields, both numeric and string, are stored as strings (ie., enclosed in quotation marks). Fields are separated from one another by either commas or carriage return-line feed combinations. The CBASIC-2 function, VAL, is used to convert numerical data stored as strings back to their numerical value for use in calculations.

The Column Descriptor File (CDF) is also a CBASIC-2 Random-access Record I/O file. This file stores information about the columns of a relation. For each column, one record is created which contains the following three fields:

(i) name of column - this could be any alphanumeric character string of arbitrary length.

(ii) column strategy - a code of 1 or 2 is used to indicate whether the data stored in a particular column is numeric or character string respectively. This is necessary for converting numeric data to be used in calculations.

(iii) field length - the maximum number of characters contained in the data of a particular column. This is equivalent to the maximum number of bytes needed to store data in that column.

The number of records in this file will equal the number of columns in a relation.

3.1.2 Differences in the Two Versions

(i) The original version of RISS took advantage of the BASIC-PLUS virtual array storage facility to store a data matrix in the RSTS/E operating system, instead of in the user-accessible computer memory. The disk file system stores data arrays and only portions of them are in core at any given time. With this facility, any element of an array in the file can be referenced randomly, for use in any BASIC-PLUS statements.

CBASIC-2 does not provide the virtual array storage facility. Hence, all arrays needed are stored in the computer memory and are restricted to a fixed number of array elements. It was necessary to convert each array element referenced in the published version of RISS to a corresponding reference to a record in one of the three files used in the CBASIC-2 version. References to zero array elements were converted to references to records with a record number equal to 1, since numbers less than 1 are not allowed for records in a file.

(ii) Random access files are stored differently in the two versions of BASIC. Whereas CBASIC-2 random access files store data in character format using the ASCII code, in BASIC-PLUS data in random access files is stored in packed binary format, thus requiring less room on the disk. Block I/O (or Record I/O) operations are performed on BASIC-PLUS random access files, necessitating several program steps that were not necessary in the CBASIC-2 version.

For instance, in creating a BASIC-PLUS random access RECORD I/O file, space must be allocated in the random buffer for each variable that will be written to the file using the FIELD statement; the LSET statement must be used to move the data into the buffer, numeric data must first be converted to strings using the BASIC-PLUS CVT%$ and CVTF$ functions. Following this, the PUT statement must be used to write data from the buffer to the disk. Similar steps using the GET statement are necessary in writing data to the Random-access Record I/O files used in the BASIC-PLUS version.

In the CBASIC-2 version of RISS, simple Record I/O operations are performed on random access files. This involves opening a file, with a fixed record length specified, and using the random form of the READ and PRINT statements. Each execution of the READ STATEMENT (used

randomly) accesses a new record. Therefore, the data in a particular field of a record is accessed by retrieving the record containing the data, reading all the record fields into a string array, and then referencing the appropriate array element. Writing data to these files was dealt with in a similar manner.

3.2 Differences in Operating System

RISS was originally implemented on a DEC PDP-11 minicomputer which used the RSTS/E operating system (Resource Sharing Time Sharing/Extended). RSTS/E is a time sharing system designed to accomodate large numbers of users. Because of this, the original version included several operations involving file manipulations and commands that were related to time-sharing functions. For example, the current - user job number was retrieved at the beginning of every program, using the PEEK statement. This number was converted to string form and used as the file type in file names used in the programs.

The CP/M operating system is a standard operating system for microcomputers. It is a disk operating system that provides file management, batch processing and program loading facilities for a single user. RSTS/E supplies a comprehensive file system in which user files can be accessed by many terminal users simultaneously.

3.3 Application Program Interface

Both the RISS data base management system and the CBASIC-2 version of RISS use multiple-line functions to implement the application-programmer interface. However, because of differences in data organization between the two versions, the function code of the CBASIC-2 functions is entirely different from that of the original RISS functions. In most cases, though, the original purpose of the functions and the function parameters have been retained. In the CBASIC-2 version it was necessary to include one other function, for the purpose of writing information to the Column Descriptor File. As a result, instead of 12 RISS functions, the application-programmer interface of the CBASIC-2 version consists of 13 functions (refer to RISS application-user interface, Chapter 2).

Insertion of these functions into the original source code of application programs is also handled differently by the two versions. In the published version, all the code of the RISS functions is collected into a single module. A special BASIC-PLUS program which acts as a pre-compiler is used to scan this module and insert only the required functions in the source code. The original source code must include at line 19999 a statement of the form:

```
! RISS 1, 2, 3 !
```

to indicate to the pre-compiler that primary functions 1, 2 and 3 are to be appended. Lines 20000 through 29990 of the original source code must be reserved for the insertion of RISS function code.

To append the function code of the CBASIC-2 RISS functions, the CBASIC %INCLUDE feature was used. This directs the compiler to compile the file specified in the INCLUDE statement, into the source code immediately following the %INCLUDE directive. The file name may contain a drive reference, but must be of type .BAS.

```
eg., %INCLUDE B:FNR1
```

The %NOLIST and %LIST directives are used together with the %INCLUDE directive to suppress listing of the function code each time a program is being compiled.

Of lesser importance, functions written in BASIC-PLUS can be defined anywhere in a program and are restricted to no more than five (5) arguments. CBASIC-2 functions must be defined prior to any reference in the program and may have any number of function arguments.

3.4 RISS Naive-user Interface

The original RISS naive-user interface level consisted of 10 command modules that allow the nonprogramming user to interact with

RISS relations. Two of these were omitted from the CBASIC-2 version. The two not included are:

1. UCREL - used for changing all lower case characters stored in a relation to upper case. This was omitted because it was not considered useful enough to warrant conversion.

2. COMREL - used for recompacting the character string storage of a RISS relation, thereby minimizing the amount of physical storage space used by a relation. This module was not necessary to the new CBASIC-2 version of RISS because recompacting was done, whenever modifications were performed on a particular relation, in the programs that made the modifications.

3.5 Program Chaining

The original RISS DBMS took advantage of the ability of BASIC-PLUS to chain from one program to another. This was done in order to control the size of a program in main memory. Execution of a CHAIN statement causes transfer of control from the program currently being executed to the program named in the CHAIN statement. The print

buffer is cleared, the return stack is reset, and any open files are closed. The chained program overlays the program which executed the CHAIN statement. The form of the CHAIN statement differs in the two versions of BASIC.

A BASIC-PLUS CHAIN statement is of the following form:

<line number> CHAIN <string> [[LINE] <number>]

where string is the name of the program to be loaded by BASIC-PLUS, compiled and executed. The line number, if specified, designates the line where BASIC-PLUS will start the program. If the line number is omitted, the chained program starts executing at its beginning.

In CBASIC-2, the CHAIN statement is of the form:

[<line number>] CHAIN <expression>

where expression must evaluate to any unambiguous file name, and specifies the program to chain to. A file type is not specified, but a file of type INT (ie., in precompiled form) must reside on the specified disk drive. The CHAIN statement causes execution to continue with the first statement in the chained program. In chaining from one program to another, the constant, code, data statement and variable areas of the main program must not be smaller than those of the chained program. CBASIC-2 provides a %CHAIN compiler directive to adjust the size of these areas, if necessary.

Examples of chaining are found in the RETREL command module, where the options of tallying, grouping and moving of records from one relation to another are performed by chaining to the programs TALLY.BAS, GROUP.BAS and MOVREL.BAS, respectively (refer to CHAPTER 2). In the MERREL command module, a third relation is created from two existing relations by chaining to the CREREL command module.

The CBASIC-2 implementation of the CHAIN statement differs from the BASIC-PLUS implementation in two ways:

(i) a COMMON statement was used to pass data from one program to another,

(ii) special code was inserted in the chained program to direct CBASIC-2 to the required portion of the program.

3.6 Error Recovery

Each of the RISS command modules in the published version attempts to recover from common data-entry errors. These errors fall into two broad areas:

1. computational errors such as division by zero
2. Input/output errors such as reading an end-of-file character (CTRL/Z) as input to an INPUT statement.

Normally when BASIC-PLUS detects an error, it prints an error message and terminates execution of the program. However, in some cases it is possible to recover from errors and continue program execution by using an ON ERROR GO TO statement. This causes the system to transfer control to a specified line number where two variables ERR and ERL are checked to determine what error occurred and to decide what action to take. ERR stores an integer number associated with an error message and ERL contains the line number of the error.

CBASIC-2 does not provide these facilities for error trapping. However, attempts were made to recover from some data-entry errors by

1. forcing the user to re-enter numeric data if it did not fall within a required range (eg., a column number must not be greater than 25).

2. using the IF END statement to process an end of file condition. When an end of file is detected on a file, control is transferred to the line number contained in the IF END statement, if an IF END statement has been executed for the file, prior to the attempted read.

3.7 Differences in CBASIC-2 Statements

Though implementations of BASIC on different computers are in many ways similar, there are some incompatibilities betwen BASIC-PLUS and

the version of BASIC used in this project (viz., CBASIC-2). The differences between CBASIC-2 statements and the BASIC-PLUS statements used in the original version of RISS are listed in the remainder of this chapter.

(i) Line numbers - since CBASIC-2 is a compiled language, line numbers are not required on every program line. They need only be present on program lines which are referenced by other program lines (GO TO, GOSUB, ON or IF statements). BASIC-PLUS is partially interpretive and hence a line number is required for every line. Since each line has a 5-byte overhead for the line number, use of line numbers in the original version of RISS was minimized by fitting several statements on one BASIC-PLUS line. This was not necessary in CBASIC-2.

(ii) Initialization of variables - Each variable has a value associated with it at all times during execution of a program written in CBASIC-2. Initially numbers are zero and strings are null strings. In the case of arrays, numeric arrays are initially set to zero and string array elements are null strings. The BASIC-PLUS version contained statements to perform this initialization. These statements were removed in the CBASIC-2 version.

(iii) Multiple Assignments - CBASIC-2 does not allow statements of the form B, C = 0. This statement in BASIC-PLUS would set the variables

B and C to zero. In the CBASIC-2 version of RISS, statements like this were rewritten as separate assignment statements viz., B = 0 : C = 0.

(iv) Identifiers - In CBASIC-2, the last character in an identifier must be a dollar sign ($) to indicate that the identifier is of type string. If the identifier ends in a percent sign (%), it represents an integer. Those identifiers not ending with a dollar sign or percent sign are of type real. BASIC-PLUS handles identifiers similarly but insists that integer constants also have a % suffix. CBASIC-2 regards the % sign after an integer constant as an invalid character and prints a warning message.

(v) IF Statement - The format of the IF statement in CBASIC-2 does not allow for nesting of IF statements. All nested IF statements used in the BASIC-PLUS version had to be rewritten as separate IF statements.

In an IF---THEN---ELSE statement in CBASIC-2, the ELSE must be followed by a statement list. BASIC-PLUS allows a line number after the ELSE. Statements of this form were converted from ELSE <line number> to ELSE GO TO <line number>.

BASIC-PLUS provides one other form of the IF statement to increase the flexibility and ease of expression within a program line.

<statement> IF <condition>.

This is illegal in CBASIC-2 and each statement of the above form was changed to the CBASIC-2 version:

IF <condition> THEN <statement>

(vi) FOR statement - CBASIC-2 lacks another statement modifier, provided by BASIC-PLUS to increase the flexibility and ease of expression within a program line. In BASIC-PLUS, it is legal to use the following to imply a FOR loop on one line:

<statement> FOR <variable> = <expression> TO <expression>

{STEP <expression>}

eg., PRINT I, SQR (I) FOR I = 1. TO 10.

Several statements of this form appeared in the BASIC-PLUS version. The CBASIC-2 equivalent was the following FOR——NEXT loop:

```
FOR I = 1.  TO 10.
PRINT I, SQR (I)
NEXT I
```

(vii) The WHILE Statement - CBASIC-2 provides a WHILE---WEND construct for looping that was not available in BASIC-PLUS. Execution of all statements between the WHILE statement and its corresponding WEND statement is repeated until the value of the expression contained in the WHILE statement is zero.

(viii) %INCLUDE directive - this causes the compiler to compile the file, specified in the INCLUDE statement, into the source immediately

following the %INCLUDE directive. The file name may contain a drive reference, and must be of type BAS.

This feature, available in CBASIC-2, was used to include the multiple line functions of the RISS application-user interface. Since the files incorporated with the %INCLUDE directive are of type BAS, they may be compiled separately. This made it easier to debug the programs using the functions, because the functions were tested individually before inclusion in the programs.

(ix) The %CHAIN directive - The %CHAIN directive is used to set the size of the main program's constant, code, data statement and variable areas. This directive must appear in the first program of a chained series to ensure that a chained program will not overwrite a portion of the data area being passed by the previous program.

The values used in the %CHAIN directive are determined by separately compiling each program in the chained series, and using the largest value for each of the four areas. The percent sign must be placed in column one.

(x) The MATCH function - is a predefined function of the form:

```
MATCH(A$, B$, I%)
```

which returns the position of the first occurrence of the substring A$ in the string B$, starting with the character position given by I%.

The function INSTR was used in BASIC-PLUS for the same purpose. The format is slightly different:

INSTR(I%, B$, A$)

This indicates a search for the substring A$ within the string B$, beginning at position I.

(xi) RIGHT$(A$,N%) - This CBASIC-2 string function returns a string consisting of the N% rightmost characters of A$.

The corresponding BASIC PLUS function is of the same form: RIGHT (A$, N%), but returns a substring of the string A$, starting from the Nth character to the last character. In converting to CBASIC-2, care had to be exercised in retrieving the correct substring.

(xii) The TAB function - is a predefined function of the form TAB (<expression>), used for positioning the output buffer pointer to the position specified by the value of the expression. Unlike BASIC-PLUS, the value of the expression in the TAB statement must be greater than or equal to one or a run-time error occurs.

(xiii) The CONSOLE statement - directs subsequent PRINT and PRINT USING statement output to the CP/M console.

The LPRINTER statement - directs subsequent PRINT and PRINT USING

statement output to the CP/M list device.

These two statements were used in the Retrieval Program (RETREL.BAS), to alternatively accept input at the terminal and send output (reports) to the printer.

These statements are not present in BASIC-PLUS, but it is possible in BASIC-PLUS to OPEN a non-file structured device, such as the terminal, for input or output. Thus, in the Retrieval program of the original RISS version, the user is permitted to enter the name of the output file for printing reports or enter a null string. If a null string was entered, then the program assigns "KB:" as the output file. This causes output to be printed on the user terminal. If an output file was specified, the information was written to this file, which was then read for printing reports.

(xiv) The CREATE Statement - This is used in CBASIC-2 to activate a new data file for reading and updating. If a file with the specified name already exists, it is deleted and a new file is created. BASIC-PLUS provides the same OPEN statement for creating new files or opening existing files.

(xv) The CLOSE Statement - deactivates one or more active files. Each file identifier listed in a CLOSE statement is released and the

associated buffer space deallocated. In CBASIC-2 it was illegal to close files that were not active. This was not so in BASIC-PLUS.

On the whole, most of the basic statements for handling files in BASIC-PLUS eg., OPEN, CLOSE, PRINT, READ, INPUT, KILL, NAME-AS etc., existed in CBASIC-2 with slight variations.

CHAPTER IV

SAMPLE DATA BASE APPLICATION

One of the objectives that data base systems attempt to accomplish is to reduce an application program's dependence on the format of the data which it processes (refer to Chapt. 1). Data independence allows application programs to be written with little regard to how data is physically stored. In this chapter, a simple application program for processing a data base of student records is presented to demonstrate the ease with which an application program can access and modify data in the data base, using the facilities of the data base management system. The application program uses a standard interface between it and the other components of the system. This interface consists of the RISS functions of the microcomputer data base management system.

4.1 Evaluating the Need for an Application Program

The converted RISS data base management system is a generalized DBMS, in that it is flexible enough to adapt to any application. This flexibility, however, makes it difficult for the non-technical user to access and modify the contents of the data base without having to consult a programmer. The naive-user interface, provided by the

converted RISS DBMS, requires the user to know a host of commands and to be able to use these commands effectively in dealing with the data base. Whenever possible a non-technical user should be provided with a facility, in which he need only specify what is wanted and the system decides how to obtain it. The application program discussed in this chapter presents such a simpler user interface, but at the expense of flexibility. Inflexibility in the application program is due to the fact that the program is desiring for a specific function and therefore all prompts for the user are strongly tied to that specific function.

4.2 Purpose of the Application Program

The main function of the application program described in this chapter is to allow the user to interactively update records in a data base, consisting of student records. Users are permitted to perform the following tasks:

(i) enter grades for assignments by section

(ii) enter marks for examinations by section

(iii) calculate and store a term mark for each student

(iv) modify grades or marks in the data base

4.3 Logical Data Base Description

A sample data base, consisting of student records, was created to demonstrate the application program. The data base consists of one relation - 'MARKS'. Fig. 4.1 gives a description of the contents of the data base. This description is the data base schema. The name of each domain (column) and relation of the data base is listed in upper case letters.

Domains: NAME NUMBER SECTION ASSIG-1 ASSIG-2
ASSIG-3 ASSIG-4 ASSIG-5 ASSIG-6 ASSIG-7
ASSIG-8 ASSIG-9 ASSIG-10 MIDTERM-1 MIDTERM-2
FINAL TERM MARK

Relations:

MARKS (Name, Number, Section, Assig-1, Assig-2, Assig-3, Assig-4, Assig-5, Assig-6, Assig-7, Assig-8, Assig-9, Assig-10, Midterm-1, Midterm-2, Final, Term Mark)

Fig. 4.1 Sample Student Data Base

Relation 'MARKS' contains information on student grades. More than one column in the relation may have the same underlying domain ie., the set of values from which entries in that column are selected. Column names

are unique within the relation.

The relation 'MARKS' is shown in Fig. 4-2, described by its table representation. Only five (5) of its columns are displayed here.

Column ——>	Name	Number	Section	Assig-1	Midterm-1
Underlying ——>	NAME	NUMBER	SECTION	ASSIG-1	MIDTERM-1
Domain	____	_____	_____	_____	_____
	ADAMS J	7906291	1	S	98
	SMITH B	7054555	2	F	76
	JONES Y	7814236	1	-	87

Fig. 4.2 Relation 'MARKS'

The rows of the table correspond to tuples of the relation, and the columns correspond to instances of particular domains of the data base. In traditional file management terms, the relation corresponds to a flat file, a tuple to a record and a column to a data field.

4.4 Description of Data Base Files

The relation MARKS is stored in three disk files: a tuple description file, a tuple file and a column descriptor file. The table below lists these files. The description given for each file includes a summary of the files' contents and uses, and any special notes that are pertinent.

Student Application Data Files

Number	Name	Description	Accessing Method
1	MARKS.TDF	This is the Tuple Descriptor File of the relation MARKS. This one-record file contains the number of tuples in the relation MARKS, the number of columns in the relation and the length of a relation tuple (in bytes). This information is used in processing the data base.	Random
2	MARKS.TF	This is the Tuple File of the relation MARKS. This file holds the name, number, section and grades or marks obtained by a student for a particular course taken during one term. At the end of the term, an overall mark for the term is entered.	Random
3	MARKS.CDF	This is the Column Descriptor File of the relation MARKS. Each record contains the name, strategy and field length for a particular column of the relation.	Random

File Layouts

File layouts describing the exact record format for each file are presented in the following pages. Each layout contains the following items:

DESCRIPTION - the name used to identify the file in this report

FILE NAME - the name that the programs use to identify the file

NO. OF RECORDS - the number of records in the file

RECORD SIZE - the number of bytes or characters each record occupies

Each field in a record is listed on a separate line. Its corresponding variable name, field description, strategy and size are shown. Strategy indicates the type of characters stored in a particular field; an alphanumeric field is indicated by a strategy of 2.

FILE LAYOUT

FILE NO.	DESCRIPTION	
1	MARKS TUPLE DESCRIPTOR FILE	
FILE NAME	**NO. OF RECORDS**	**RECORD SIZE**
MARKS.TDF	1	50

MISCELLANEOUS COMMENTS

VARIABLE	FIELD DESCRIPTION	STRATEGY	MAX. SIZE
R2% (1)	Number of Tuples	1	
R3% (1)	Number of Columns	1	2
R4% (1)	Length of Tuple	1	512

FILE LAYOUT

FILE NO.	DESCRIPTION	
2	MARKS TUPLE FILE	
FILE NAME	NO. OF RECORDS	RECORD SIZE
MARKS.TF		122

MISCELLANEOUS COMMENTS

Any field can be used as a key.
The tuple file contains 1 record for each student.

VARIABLE	FIELD DESCRIPTION	STRATEGY	MAX. SIZE
T$ (1)	STUDENT NAME	2	40
T$ (2)	STUDENT NUMBER	1	7
T$ (3)	SECTION NUMBER	1	1
T$ (4)	GRADE FOR ASSIGNMENT 1	2	1
T$ (5)	GRADE FOR ASSIGNMENT 2	2	1
T$ (6)	GRADE FOR ASSIGNMENT 3	2	1
T$ (7)	GRADE FOR ASSIGNMENT 4	2	1
T$ (8)	GRADE FOR ASSIGNMENT 5	2	1
T$ (9)	GRADE FOR ASSIGNMENT 6	2	1
T$ (10)	GRADE FOR ASSIGNMENT 7	2	1
T$ (11)	GRADE FOR ASSIGNMENT 8	2	1
T$ (12)	GRADE FOR ASSIGNMENT 9	2	1
T$ (13)	GRADE FOR ASSIGNMENT 10	2	1
T$ (14)	MARK FOR FIRST MIDTERM	1	3
T$ (15)	MARK FOR SECOND MIDTERM	1	3
T$ (16)	MARK FOR FINAL EXAM	1	3
T$ (17)	TERM MARK	1	5

FILE LAYOUT

FILE NO.	DECRIPTION	
3	MARKS COLUMN DESCRIPTOR FILE	
FILE NAME	NO. OF RECORDS	RECORD SIZE
MARKS.CDF	17	50

MISCELLANEOUS COMMENTS

This file contains 1 record for each column in the relation 'MARKS'.

VARIABLE	FIELD DESCRIPTOR	STRATEGY	MAX. SIZE
N$	COLUMN NAME	2	VARIABLE
S$	COLUMN STRATEGY	1	1
F$	COLUMN FIELD LENGTH	1	VARIABLE

4.5 Building the Data Base Files

Each data base relation is created by naming the relation and its constituent columns and specifying the storage strategy and size of each column. The microcomputer DBMS provides a facility for creating relations viz., the command module CREREL. This module requests schema definitions from the user and creates the three files for the relation 'MARKS'. Besides entering the number of columns in the relation, the following information is entered under the following headings:

NAME	STRATEGY	FIELD LENGTH
NAME	2	40
NUMBER	1	7
SECTION	1	1
ASSIG-1	2	1
ASSIG-2	2	1
ASSIG-3	2	1
ASSIG-4	2	1
ASSIG-5	2	1
ASSIG-6	2	1
ASSIG-7	2	1
ASSIG-8	2	1
ASSIG-9	2	1
ASSIG-10	2	1
MIDTERM-1	1	3
MIDTERM-2	1	3
FINAL	1	3
TERM MARK	1	5

The above information is stored in the column descriptor file, each row of the table being stored in one record. As a result, this file contains 17 records.

At the end of execution of program CREREL, the tuple descriptor file contains the number zero in the first field of its record, since at this time there are no tuples in the relation. The second field stores the number 17 (the number of columns in the relation). As field lengths are entered for each column, their values are summed and a relation tuple length is calculated. This value (equal to 122) is stored in the third field of the record.

The tuple file is an empty data file - there are as yet no tuples in the relation.

4.6 Loading the Data Base

Tuples are added to the data base by means of the command module EDREL (refer to Chapt 2.). This is one of the modules provided by the naive-user interface of the converted data base management system. Using the insert command (I), a tuple is added to the relation for each student enrolled in a particular course. The program prints the names of each column of the tuple and the user enters the values of data to be stored in the various columns. For the purpose of the application

discussed in this chapter, only the columns specified by name, number and section number are filled with true data; the remaining columns are filled with null data. These columns represent grades for assignments and marks for examinations. The application program will be used to replace these null strings with actual grades/marks as they are obtained.

4.7 Application Program Description

The application program is used to update a data base consisting of student records. It also demonstrates the use of the RISS functions in writing application programs to access and modify data in the data base. The primary functions used are FNR1$ for initializing the relation 'MARKS', FNR2$ for retrieving data from a specified column of the relation, and FNR3$ for storing data in a specified column of the relation. The programming language used is CBASIC-2, the same language used in developing the RISS functions.

4.7.1 Input to Application Program

The application program STUDENT (file name is STUDENT.BAS) is run in interactive mode, hence, all input is supplied through the terminal. In general, a question is asked by the program which prompts the user for an answer. The information that is required from the user, along

with the range of values for each piece of information is listed in the following table:

Input	Range of Values
Assignment Number	1 - 10
Examination Number	1 - 3
Section Number	1 - 6
Student Number	No restriction
Weight on Assignments	0 - 1
Weight on First Midterm	0 - 1
Weight on Second Midterm	0 - 1
Weight on Final Exam	0 - 1

All input data are checked by the application program to ensure that their values fall within the desired range. No validity checks are performed on the student number. However, if the user enters a student number that does not exist in the data base, the program after searching for the number, prints the student number and reports that it was not found.

4.7.2 Program Narrative

The program may be divided into five sections, corresponding to the different functions it performs. These sections are listed here in the same order in which they appear in the program.

Section 1 - Enter grades for assignments by class section

Section 2 - Enter marks obtained on examinations by class section

Section 3 - Modify grades or marks in a student record

Section 4 - Calculate and store an overall term mark for each student in the data base

Section 5 - Quit operations

These five tasks along with their identifying numbers are contained in a section of code consisting of CBASIC-2 PRINT statements. At the beginning of program execution the tasks, with their numbers, are displayed on the terminal and the user is prompted to select the task he wishes to perform by entering the identifying number. At the end of every operation selected by the user, these tasks are displayed on the screen so that the user can easily specify what he wants to do next.

The RISS primary function, FNR1$, is referenced at the beginning of the program to initialize the relation 'MARKS'. Since there is only one relation in this application, a logical relation number of 1 is used in the functions' parameter list. This causes the files MARKS.TDF, MARKS.TF and MARKS.CDF to be opened with file identification numbers 10, 11 and 12 respectively. Information from the TDF are then made core resident, so that:

R2% (1) = number of student records

R3% (1) = number of fields in a record

R4% (1) = length of a record in bytes

When this is done, the program prints the five user options and prompts the user to select one, by number. On specification of this number, the program transfers to one of five sections of code, as described below:

Section 1 - Enter Assignment Marks

To enter assignment marks, the user is first requested to input the assignment number. A check is made to ensure that this number is a valid assignment number ie., falls within the desired range. If the assignment number is not valid, a report is printed to this effect and the user is forced to re-enter an assignment number. If the number entered was valid, the program then requests the number of the section that the student is in. This number is also limited to a specified range and a check is made to ensure that it falls within this range. The program then enters a loop in which only those records belonging to the particular section are retrieved for processing. This is achieved by checking the third field of the tuple file for the section number. For each record belonging to the particular section, the name and student number is displayed on the screen, and a grade is requested. This grade is then stored in the appropriate column. The number of

this column is evaluated to three (3) more than the number given for the assignment.

As the loop is executed, the record numbers for students in this particular section are stored in an integer array T%. When all the grades are entered, this array is used for checking that the entries made in the data base are correct (without having to search the whole data base again). Checking of entries is optional. If the user does not wish to do any checking, he is allowed to exit from this section of the program and select another task. Otherwise, the name, student number and grade entered for each student in the section is displayed on the terminal and for each student displayed the user may choose to change the grade or leave it unchanged. If the user wishes to change the grade, he is prompted to enter a new grade for the student and this new grade is then stored. When all the processing for the section is completed, a report is printed to notify the user that the assignment marks have been entered. The program then exits from this section to display the user option on the terminal and prompt the user to enter an option, by number.

Section 2 - Enter Examination Marks

Entry of examination marks involves program steps similar to those used in entering assignment grades. Instead of an assignment number

the user is requested to input an examination number. This number is used to calculate the number of the column in which the examination mark would be stored. The user is also required to specify a section number, so that only those records in a particular section are made available to the user for entering marks. The name and number of each student in the section is displayed on the terminal and the user is prompted for a mark. The program then stores this mark in the correct column. The number of this column evaluates to thirteen (13) more than the examination number specified. The user may choose to check entries if he wishes to do so.

Section 3 - Modify Student Marks

This section is used to modify assignment grades or examination marks. The user is first requested to select which of the two tasks he would like to perform. He is then asked to enter an assignment number or an examination number, depending on which task he selected. A range check is performed on the number specified. This number is then used to calculate the number of the column in which the data is to be modified. The user must then specify the number of the student, whose marks are to be changed. On specification of this number, the records in the data base are searched for this number. If a record with this

student number cannot be found, a report is printed to this effect. Otherwise, the name and number of the student, along with the grade/mark made for the particular assignment/examination are displayed on the terminal. The program then asks the user to enter the new mark, which is then stored in the appropriate column. This procedure loops until the user specifies that he no longer wishes to make any changes.

Section 4 - Calculate and Store Term Marks

This section of code calculates an overall mark for the term (term mark) and stores it in the last (17th) column of the file MARKS.TF. The user must first enter the following information:

(i) weight on assignemnt - WA

(ii) weight on first mid-term examination -W1

(iii) weight on second mid-term examination -W2

(iv) weight on final exam - W3

For each student, the program calculates the total mark made on assignments, according to a specified rule: "If the grade obtained for an assignment is an 'S', the student gets a mark, otherwise he gets a mark of zero". A loop is used to total the amount made on the 10 assignments.

The overall mark for the term is calculated as follows:

```
Term Mark = (WA * total on assignments +
            W1 * mark on first mid-term +
            W2 * mark on second mid-term +
            W3 * mark on final exam)
```

It was required that the term mark be stored with one digit after the decimal point, if the mark obtained was not a whole number. This was achieved by converting the number to a string, and employing the user-defined string function (FNTR$), for obtaining the required substring starting from the first digit in the number up to the digit after the decimal point. The digit after the decimal point was also rounded by this function. The term mark was then stored as a string, by the program. For the benefit of the user, as each term mark is calculated, it is displayed on the terminal along with the corresponding student number. When marks have been calculated and entered for all students in a given section, a report is printed indicating the end of operations in this section.

Section 5 - Quit operations

This section allows the user to terminate operations on the data base and exit to the CP/M monitor. Files that were active in the

program are closed and the following prompt appears on the screen:

A>

4.7.3 Output of Application Program

All output from the application program STUDENT is displayed on the terminal. Output is in the form of two kinds of reports.

(i) Exception reports - This report prints all bad transactions with appropriate error messages. Examples of error messages are:

INVALID ASSIGNMENT NUMBER - 25

INVALID EXAMINATION NUMBER - 10

(ii) Update reports - This kind of report is printed at the end of a successful operation on the data base. Examples are:

EXAMINATION MARKS HAVE BEEN ENTERED.

TERM MARKS HAVE BEEN ENTERED.

The emphasis in this program was not on reporting. The naive user interface of the converted RISS DBMS provides a good retrieval program, which may be used to send formatted reports, on the updated data base, to the printer or screen.

4.7.4 Using the Application Program

A. To start the program STUDENT.BAS

1. Turn the power on (if necessary).

2. Place the correct disk(s) in the correct drive(s).

This prompt should appear:
A>

3. Type the underlined command with the correct password.
A> CRUN2 STUDENT password

4. Choose desired option

B. To end program STUDENT.BAS

1. Choose option 5. This prompt appears: A>

2. Remove disk(s) from drive(s).

3. Turn the power off.

C. Points to note in using the program

1. Each data entry must be accompanied by a carriage return (hit the key marked RETURN).

2. If mistakes are made in entering assignment grades or examination marks, two things can be done:

(i) record the student numbers whose marks were entered incorrectly and then use option 3 to modify only these marks.

(ii) while still in the particular option for entering grades/marks, respond with a Y to the prompt 'DO YOU WISH TO CHECK ENTRIES >' The program will go through all the records in the section and allow the user to change the erroneous ones.

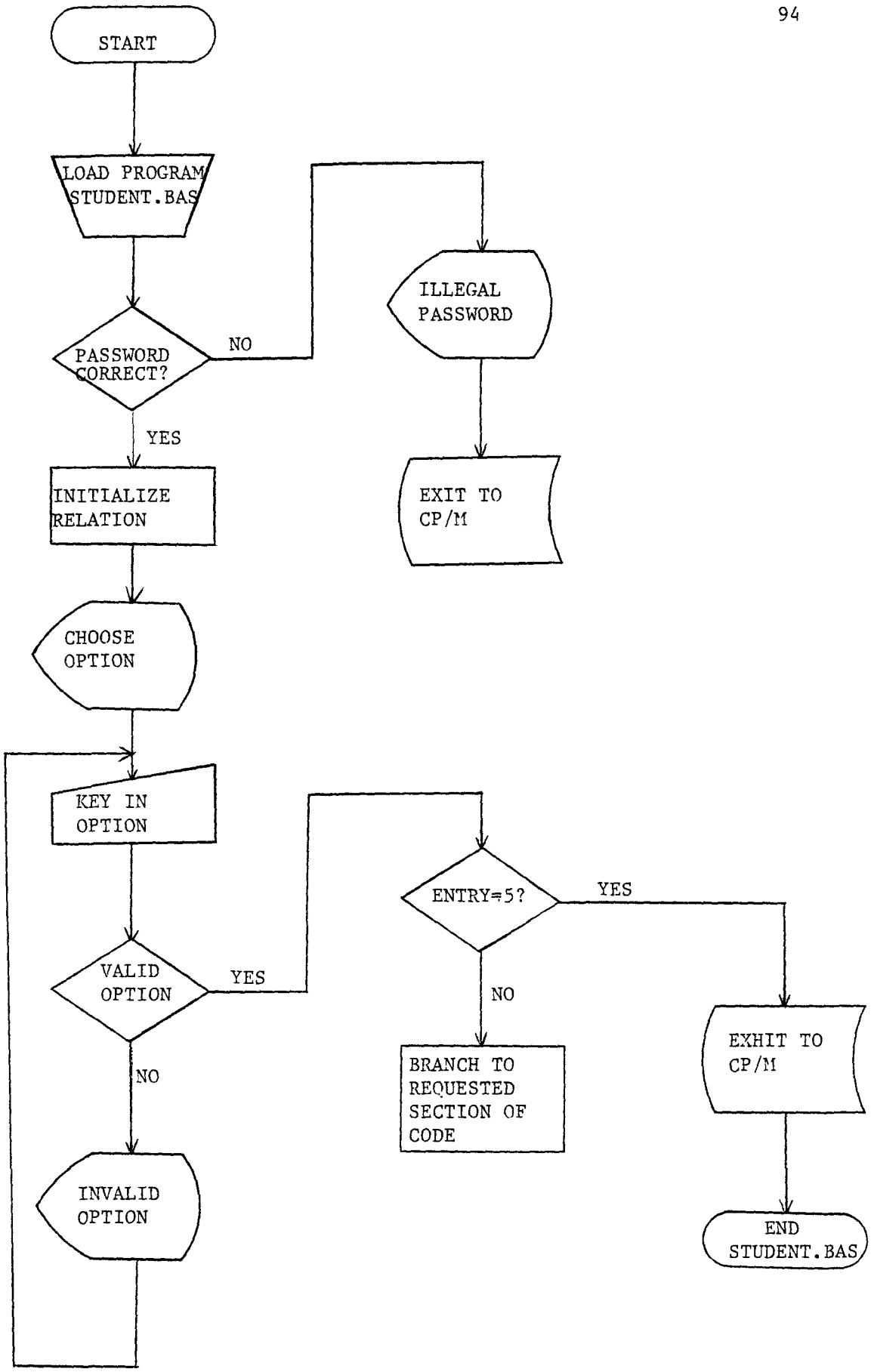
3. If a mistake is made in selecting the task to perform, the user may exit from that option by typing N when the prompt 'Confirm Option >' appears on the screen.

4. All key-in mode entries are tested for validity. This test includes a range check for numeric entries. When some sort of unacceptable entry is discovered by this test, the user must re-enter the valid data.

5. To use this program, the user must correctly enter a password before he can proceed with the task selection. This protects the student data base from accidental or malicious tampering.

4.8 Privacy Controls

The data base is stored in direct access storage and accessed directly by an application using standard operating system access methods. Since the contents of the data base must be confidential for obvious reasons, the data base will be accessible only to authorized users. This is accomplished by the use of a password. A user desiring access to the student data base must supply a password before the access is allowed.



FLOWCHART OF APPLICATION PROGRAM

CHAPTER V

CONCLUSION

The original goal of this project was to convert the RISS data base management system, originally implemented on a minicomputer, for use on a microcomputer. This goal has been successfully achieved. The converted DBMS, written in CBASIC-2, has been installed, accepted and is operational on a Dynabyte microcomputer using the CP/M operating system. The data base management system may be used to support one or more data bases on any microcomputer which uses a CP/M operating system and handles CBASIC-2.

The secondary objective of the project has also been accomplished. As seen in Chapter 4, the converted DBMS was employed in creating a data base of student records for the Faculty of Business on their Dynabyte microcomputer. The module CREREL successfully created the single relation in the student data base. The EDREL module was used to build the data base and also for examining, modifying and deleting tuples in the relation 'MARKS' of the student data base. For retrieving data from the data base, the module RETREL was invoked. The AND (A) and OR (O) commands of RETREL facilitated querying the data base and retrieving selected subsets of the data base. RETREL was also used in producing formatted reports on the screen or at the printer. Thus these

modules have been sufficiently tested.

The remaining modules of the data base maintenance package ie., MERREL, COLREL, COPREL, SORREL AND DELREL were separately tested and are basically error-free.

The converted data base management system has a high degree of data independence in that physical storage is separate from the logic of applications using the data. This aids flexibility and ease of change in a dynamic environment. Adding a new relation to the data base or a new column to a relation does not impact existing application programs. Since the data base system has been designed in a modular and structured manner, it is easy to expand it to include more application programs while maintaining data independence.

5.1 Disadvantages of the Relational Model

The relational model is much simpler from the viewpoint of the user. The number of basic constructs is one, namely the relation (or table) itself; all information in the data base is represented using just this one construct. Relations are also easy to manipulate. Adding a new relation to a data base or a new column to a relation does not impact existing application programs.

Unfortunately, the price that must be paid for simplicity to the user is complexity to the data base system. It is difficult to design an efficient, effective relational data base system. For large data

bases, where the cost of storage space and computer time dominate the total cost of implementing the data base, the network and hierarchical models are far superior. As an example, the most simple-minded implementation technique used in relational data base systems is an exhaustive search of the data base. For handling single tuple variable requests, this technique has an execution time that is proportional to the number of tuples in the relation and the overhead incurred in large data bases is unacceptable. However, the type of data base implemented on a microcomputer is normally physically small, making a relational data base structure a suitable choice for such systems.

5.2 Limitations of CBASIC-2

Although BASIC is one of the easiest languages to learn and use, certain limitations of the language made the experience of programming in CBASIC-2 an unhappy one.

1. It does not have a very rich set of control structures; (for example, nesting of IF statements is not allowed).
2. String handling capabilities are minimal.
3. Input/Output capabilities are incomplete.
4. Execution of a CHAIN statement causes transfer of control from the program being executed to the beginning of the chained program. Extra code had to be written to effect the transfer to a desired portion of the chained program.
5. Routines for error trapping are not available.

6. The syntax of the language does not allow building complex block structures, and the result is a limitation of the complexity of programs which can be written.

7. CBASIC-2 uses a compiler for language translation. Hence, any single change to a completed program required that the entire program be recompiled. This proved to be very time-consuming during program debugging.

5.3 Limitations of Microcomputers

The advantages of a microcomputer over a traditional minicomputer are:

1. lower cost (perhaps 10 times less),
2. small size,
3. lower power consumption,
4. availability of new low cost peripherals.

Its main limitations compared to a minicomputer are:

1. lesser processing power - the use of an 8-bit microprocessor results in a slower execution speed. The processing efficiency of the microcmputer DBMS could be improved by using a hard disk, as opposed to floppy disks. Access time of floppy disks is slow because of repeated access to the files of the RISS system. The original implementation of RISS on a minicomputer which used a hard disk was efficient. Hence, the microcomputer data base management system is expected to be more

efficient if files are stored on hard disk.

2. less software - because microcomputers are less technically advanced than minicomputers, their software libraries are smaller. However, this situation should improve over the next few years.

5.4 Future Enhancements

During the development of the microcomputer data base management system, several areas of improvement were noticed. These were not implemented here, either because of time restrictions or because they were beyond the scope of the initial project. Briefly, these improvements are :

1. Data integrity - there is need for a greater degree of data integrity in the data base. This implies a need for several constraints on the data. Constraints may be of two types:

(i) validity constraints eg., a data field may be allowed to contain only values whose type and format exactly match the characteristics declared, or the value of a particular field may be required to fall within a particular range.

(ii) consistency constraints; for example, values of a data field may be required to appear also as values in some field in another tuple.

In this implementation , the only constraint imposed on the data is the specification of the storage strategy, indicating that the data of a particular column is either numeric or character string. The data

base management system should provide for data base integrity through control of updates to assure validity of the data base. Validation procedures could be written which would examine the input data when any storage operation is attempted.

2. Privacy controls on the data base - the data base management system should provide security capabilities to assist in assuring that information is available only to those entitled to it and that only authorized persons may update the data base. At present, no such privacy controls exist on the converted DBMS.

3. Crash protection and recovery - to protect the data base against accidental loss, facilities to reconstruct the data base after a hardware or software error should be provided. This may be accomplished by the creation of an audit trail (or log) and through special programs designed to recreate a damaged data base.

4. Query reporting - a further extension of the query reporting program could be made to improve the interactive capability of the system, over and above the so-called 'naive-user' interface contained within the current RISS implementation. Depending on future needs, additional special report programs can be written to query the data base.

5.5 Concluding Remarks

It is quite clear that for small business applications, a very useful DBMS could be based on a microcomputer. This gives a cheap, light and portable system yielding results on the spot. The data base task might once have been thought best solved by time shared access to a large computer. However, economic considerations in many cases will show the advantages of microcomputer implementations of data base management systems.

APPENDIX A

SAMPLE DIALOGUE FOR NAIVE-USER INTERFACE OF MICROCOMPUTER DBMS

CREREL - CREATING RELATIONS

```
A>CRUN2 B:CREREL

CRUN VER 2.05

NAME THE RELATION TO BE CREATED > MARKS
CONFIRM CREATION OF RELATION MARKS >? Y
NUMBER OF COLUMNS (ORDER) IN THE RELATION > 4

COLUMN 1
     COLUMN NAME > NAME
     STRATEGY  > 2
     FIELD LENGTH > 40

COLUMN 2
     COLUMN NAME > NUMBER
     STRATEGY  >  2
     FIELD LENGTH > 7

COLUMN 3
     COLUMN NAME > SECTION
     STRATEGY > 1
     FIELD LENGTH > 1

COLUMN 4
     COLUMN NAME >  MARK
     STRATEGY > 1
     FIELD LENGTH > 3

ANY CORRECTIONS > Y
COLUMN TO CORRECT > 2
CHANGE COLUMN NAME (Y OR N) > N
CHANGE COLUMN  STRATEGY (Y OR N) > Y
     NEW  STRATEGY > 1
CHANGE FIELD LENGTH (Y OR N) > N

ANY CORRECTIONS > N

RELATION MARKS HAS BEEN  CREATED
```

EDREL - EDITING RELATIONS

```
A> CRUN2 B:EDREL

CRUN VER 2.05

RELATION TO EDIT > MARKS
COLUMN NAMES (Y OR N)? > Y
HOW MANY COLUMNS TO DISPLAY? > 1
COLUMN# 1 >? 1
RELATION MARKS :  0  RECORDS,  4 COLUMNS.

*  ?
RELATION: MARKS
RECORD: 0

*  I
1 .> NAME? ADAMS B
2 .> NUMBER? 7005545
3 .> SECTION? 1
4 .> MARK? 78

*  B
1. > NAME? BENTLEY F
2. > NUMBER? 7915145
3. > SECTION? 2
4. > MARK? 83

*  ?
RELATION: MARKS
RECORD: 2

*  B
1. > NAME? COREY J
2. > NUMBER? 7302789
3. > SECTION? 1
4. > MARK? 88
```

```
* -2
1> ADAMS B

* P3

1.NAME >ADAMS B    2.NUMBER >7005545    3.SECTION >1  4.MARK >78

1.NAME >BENTLEY F    2.NUMBER >7915145    3.SECTION >2   4.MARK >83

1.NAME >COREY J    2.NUMBER >7302789    3.SECTION >1    4.MARK  >88

* -1
1 > BENTLEY F

* E
COLUMN > 2
7915145

* V
COLUMN > 4
83
NEW STRING > 85

* P1

1.NAME >BENTLEY F    2.NUMBER >7915145    3.SECTION >2    4.MARK >85

* B
 1 .> NAME? HARPER A
 2 .> NUMBER? 7924236
 3 .> SECTION? 1
 4 .> MARK? 75

* ?
RELATION: MARKS
RECORD: 4

* -9
 1 > ADAMS B
```

```
* P4

1.NAME >ADAMS B   2.NUMBER >7005545   3.SECTION >1   4.MARK >78

1.NAME >BENTLEY F   2.NUMBER >7915145   3.SECTION >2   4.MARK >85

1.NAME >COREY J   2.NUMBER >7302789   3.SECTION >1   4.MARK >88

1.NAME >HARPER A   2.NUMBER >7924236   3.SECTION >1   4.MARK >75

* ?
RELATION: MARKS
RECORD: 4

* L
COLUMN > 4
SEARCH STRING > 2
STRING NOT FOUND

* L
COLUMN > 3
SEARCH STRING > 2
2
 1 > BENTLEY F

* ?
RELATION: MARKS
RECORD: 2

* S
COLUMN > 2
SEARCH STRING > 1
7915145
NEW STRING > 2
 1 > BENTLEY F

* P1

 1.NAME >BENTLEY F   2.NUMBER >7925245   3.SECTION >2   4.MARK >85
```

```
* B
 1 .> NAME? JEROME W
 2 .> NUMBER? 7305758
 3 .> SECTION? 2
 4 .> MARK? 64

* ?
RELATION: MARKS
RECORD: 5

* -99
 1 > ADAMS B

* P99

 1.NAME >ADAMS B    2.NUMBER >7005545    3.SECTION >1    4.MARK >78

 1.NAME >BENTLEY F    2.NUMBER >7925245    3.SECTION >2    4.MARK >85

 1.NAME >COREY J    2.NUMBER >7302789    3.SECTION >1    4.MARK >88

 1.NAME >HARPER A    2.NUMBER >7924236    3.SECTION >1    4.MARK >75

 1.NAME >JEROME W    2.NUMBER >7305758    3.SECTION >2    4.MARK  >64

* Q

A>
```

RETREL - RETRIEVING DATA FROM RELATIONS

```
A> CRUN2  B:RETREL
CRUN VER 2.05
RELATION TO RETRIEVE DATA FROM > MARKS
OUTPUT TO SCREEN OR PRINTER (S OR P)? > S
RELATION MARKS:  5 RECORDS,  4 COLUMNS.

*  C
 1  = NAME
 2  = NUMBER
 3  = SECTION
 4  = MARK

*  P
NUMBER OF COLUMNS > 5
COLUMN  1 >? 0
COLUMN  2 >? 1
COLUMN  3 >? 2
COLUMN  4 >? 3
COLUMN  5 >? 4
DELIMITER (MAY BE NULL) >
DO YOU WANT ALIGNED COLUMNS? > Y
POSITION TO BEGIN COLUMN  1 >? 1
POSITION TO BEGIN COLUMN  2 >? 10
POSITION TO BEGIN COLUMN  3 >? 20
POSITION TO BEGIN COLUMN  4 >? 30
POSITION TO BEGIN COLUMN  5 >? 40

         NAME       NUMBER     SECTION       MARK

 1       ADAMS B    7005545    1             78
 2       BENTLEY F  7925245    2             85
 3       COREY J    7302789    1             88
 4       HARPER A   7924236    1             75
 5       JEROME W   7305758    2             64
```

```
*  A
COLUMN > 3
<, >, OR = (WITCH ONE) > =
EXACT MATCH REQUIRED?> Y
VALUE > 1

*  P
NUMBER OF COLUMNS > 3
COLUMN  1 >? 1
COLUMN  2 >? 2
COLUMN  3 >? 3
DELIMETER (MAY BE NULL) >
DO YOU WANT ALIGNED COLUMNS? > Y
POSITION TO BEGIN COLUMN  1 >? 10
POSITION TO BEGIN COLUMN  2 >? 20
POSITION TO BEGIN COLUMN  3 >? 30

         NAME       NUMBER     SECTION

         ADAMS B    7005545    1
                    7302789    1
```

```
*  A
COLUMN > 3
<, >, OR = (WITCH ONE) > =
EXACT MATCH REQUIRED?> Y
VALUE > 1

*  P
NUMBER OF COLUMNS > 3
COLUMN  1 >? 1
COLUMN  2 >? 2
COLUMN  3 >? 3
DELIMETER (MAY BE NULL) >
DO YOU WANT ALIGNED COLUMNS? > Y
POSITION TO BEGIN COLUMN  1 >? 10
POSITION TO BEGIN COLUMN  2 >? 20
POSITION TO BEGIN COLUMN  3 >? 30

         NAME       NUMBER    SECTION

         ADAMS B    7005545   1
         COREY J    7302789   1
         HARPER A   7924236   1

*  I
RELATION MARKS:  5 RECORDS,  4 COLUMNS

*  T
COLUMN TO TALLY > 3

SECTION

1                          3         (60.00%)
2                          2         (40.00%)
#MISSING DATA#             0         ( 0.00%)

*  S
COLUMN > 4
DO YOU WANT STANDARD DEVIATION? > Y
```

```
MARK

    RECORDS      =      5
    MEAN         =      78.00
    MEDIAN       =      78.00
    ST DEV.      =       8.41

*  G
COLUMN TO GROUP > 4
NUMBER OF GROUPS > 3

CUT OFF 1 >? 70

CUT OFF 2 >? 80

MARK
< 70                              1        (20.00%)
< 80                              2        (40.00%)
>=                                2        (40.00%)

*  O
COLUMN > 4
<, >, OR = (WHICH ONE) > >
VALUE > 70

*  P
NUMBER OF COLUMNS > 2
COLUMN 1 >? 1
COLUMN 2 >? 2
DELTMITER (MAY BE NULL) >
DO YOU WANT ALIGNED COLUMNS? > Y
POSITION TO BEGIN COLUMN  1 >? 10
POSITION TO BEGIN COLUMN  2 >? 20

          NAME       MARK

          ADAMS B    78
          BENTLEY F  85
          COREY J    88
```

```
         HARPER A  75

*  M
RELATION TO MOVE RECORDS TO > MARKSBCK
ERASE RECORDS AFTER MOVING? > N
A '#' WILL APPEAR FOR EACH RECORD MOVED.
####

MOVED FROM RELATION MARKS TO RELATION MARKSBCK: 4 RECORDS.

A>
```

COPREL - COPYING RELATIONS

```
A> CRUN2  B:COPREL

CRUN VER 2.05

RELATION TO COPY > MARKS
NEW RELATION > MARKSBCK
CONFIRM COPY (Y OR N) > Y

RELATION MARKS HAS BEEN COPIED TO RELATION MARKSBCK
```

DELREL - DELETING RELATIONS

```
A> CRUN2  B:DELREL

CRUN VER 2.05

RELATION TO DELETE > MARKSBCK
```

```
CONFIRM DELETION > Y

RELATION MARKSBCK HAS BEEN DELETED
```

COLREL - MAINTAINING RELATION COLUMNS

```
A> CRUN2  B:COLREL

CRUN VER 2.05

RELATION TO MODIFY > MARKS2

DO YOU WISH TO:
       S) SET A COLUMN NAME.
       D) DELETE A COLUMN.
       A) ADD A COLUMN.

OPTION (S, D, OR, A) > S

COLUMN TO RENAME > 1
NEW COLUMN NAME >  NAMES
COLUMN ONE IS NOW NAMED NAMES

ANY MORE OPERATIONS ON THIS RELATION (Y OR N) > Y

DO YOU WISH TO:
       S) SET A COLUMN NAME.
       D) DELETE A COLUMN.
       A) ADD A COLUMN.

OPTION (S, D, OR A) > D

COLUMN TO DELETE > 4

COLUMN 4 HAS BEEN DELETED.
```

```
ANY MORE OPERATIONS ON THIS RELATION (Y OR N) > Y

DO YOU WISH TO:
       S) SET A COLUMN NAME.
       D) DELETE A COLUMN.
       A) ADD A COLUMN.

OPTION (S, D, OR A) > A

COLUMN AFTER WHICH TO ADD THE NEW COLUMN > 4
NEW COLUMN NAME > GRADE
NEW COLUMN STATEGY > 2
NEW FIELD LENGTH > 1

THE NEW COLUMN HAS BEEN ADDED AFTER COLUMN 2

ANY MORE OPERATIONS ON THIS RELATION (Y OR N) > N

A>
```

MERREL - MERGING RELATIONS

```
A> CRUN2  B:MERREL

CRUN VER 2.05
FIRST RELATION > MARKS
SECOND RELATION > MARKS2
THIRD RELATION > MERGE

NUMBER OF COMPARISON COLUMNS PAIRS > 2
COLUMN 1
      COLUMN NUMBER IN THE FIRST RELATION > 1
      COLUMN NUMBER IN THE SECOND RELATION > 2
COLUMN 2
      COLUMN NUMBER IN THE FIRST RELATION > 2
      COLUMN NUMBER IN THE SECOND RELATION > 4
```

```
2 RECORDS RESULTED FROM THE MERGRE


                SORREL - SORTING RELATIONS

A> CRUN2  B:SORREL

CRUN VER 2.05

RELATION TO SORT 7 MARKS

COLUMN TO SORT ON > 5

CONFIRM SORT > Y

RELATION MARKS HAS BEEN SORTED.
```

Note that in this implementation the data base management system resides on a disk in drive B. Also note that in the sample dialogues, the user responses are underlined.

APPENDIX B

LISTING OF SAMPLE APPLICATION PROGRAM

```
      REM    ********************************************************
      REM    *  THIS IS A SAMPLE APPLICATION PROGRAM TO ILLUSTRATE  *
      REM    *  THE USE OF THE RISS FUNCTIONS. THE PROGRAM  ALLOWS  *
      REM    *  THE USER INTERACTIVELY TO ADD OR MODIFY  DATA IN A  *
      REM    *  DATA BASE OF STUDENT RECORDS                        *
      REM    ********************************************************

      PASSWORD$="PASS"
      IF COMMAND$<>PASSWORD$ THEN PRINT "ILLEGAL PASSWORD":STOP
%INCLUDE FNR5%
%INCLUDE FNR5
%INCLUDE FNR7
%INCLUDE FNR1
%INCLUDE FNR2
%INCLUDE FNR3
      DIM T%(200),R2%(3),R3%(3),R4%(3),R7%(3)
      V$=FNR1$("MARKS",1)
10    PRINT
      PRINT "DO YOU WISH TO: "
      PRINT "     1) ENTER ASSIGNMENT MARKS "
      PRINT "     2) ENTER EXAMINATION MARKS "
      PRINT "     3) MODIFY STUDENT MARKS "
      PRINT "     4) CALCULATE TERM MARKS "
      PRINT "     5) QUIT "
      PRINT
20    INPUT "TYPE OPTION (1,2,3,4 OR 5) >";OP%
      IF OP%<1 OR OP%>5 THEN 20
      PRINT
      ON OP% GO TO 100,200,300,400,32010

100   PRINT "OPTION - ENTER ASSIGNMENT MARKS"
      INPUT "CONFIRM OPTION (Y OR N) >";Y$
      IF ASC(Y$) <> 89 THEN 10
      INPUT "ENTER ASSIGNMENT NUMBER >";AN%
      IF AN%<1 OR AN%>10 THEN 100
      C%=AN%+3
      GOSUB 1001
      PRINT
      INPUT "DO YOU WISH TO CHECK ENTRIES >";Y$
      IF ASC(Y$) <> 89 THEN 120
      GOSUB 2001
120   PRINT
      PRINT "ASSIGNMENT MARKS HAVE BEEN ENTERED."
      PRINT
      GO TO 10

200   PRINT "OPTION - ENTER EXAMINATION MARKS"
      INPUT "CONFIRM OPTION (Y OR N) >";Y$
      IF ASC(Y$) <> 89 THEN 10
      INPUT "ENTER EXAM NUMBER (1,2 OR 3) >";EN%
```

```
      IF EN%<1 OR EN%>3 THEN 200
      C%=EN%+13
      GOSUB 1001
      PRINT
      INPUT "DO YOU WISH TO CHECK ENTRIES >";Y$
      IF ASC(Y$)<>89 THEN 220
      GOSUB 2001
220   PRINT
      PRINT "EXAMINATION MARKS HAVE BEEN ENTERED."
      PRINT
      GO TO 10

300   PRINT "OPTION - MODIFY STUDENT MARKS"
      INPUT "CONFIRM OPTION (Y OR N) >";Y$
      IF ASC(Y$) <> 89 THEN 10
      INPUT "CHANGE (1)ASSIGNMENT MARKS OR (2)EXAM MARKS (1 OR 2) >";M%
      IF M%<1 OR M%>2 THEN 300
      IF M%=1 THEN\
         INPUT "ENTER ASSIGNMENT NUMBER >";AN%:\
         GO TO 310\
      ELSE\
         INPUT "ENTER EXAMINATION NUMBER >";EN%:\
         GO TO 315
310   IF AN%<1 OR AN%>10 THEN\
         PRINT " INVALID ASSIGNMENT NUMBER - ";AN%:\
         PRINT:GO TO 300
      C%=AN%+3:GO TO 320
315   IF EN%<1 OR EN%>3 THEN\
         PRINT " INVALID EXAMINATION NUMBER - ";EN%:\
         PRINT:GO TO 300
      C%=EN%+13
320   INPUT "ENTER STUDENT NUMBER >";ID$
      FOR I%=1 TO R2%(1)
         V$=FNR2$(I%,2,1)
         IF V$=ID$ THEN\
            PRINT:\
            PRINT FNR2$(I%,1,1);"  ";FNR2$(I%,2,1);" _ ";FNR2$(I%,C%,1):\
            INPUT "  ENTER NEW MARK >";A$:\
            V$=FNR3$(I%,C%,1,A$):\
            GO TO 330
      NEXT I%
      PRINT ID$;" - STUDENT NUMBER NOT FOUND"
330   INPUT "  ANY MORE CHANGES? >";Y$
      PRINT
      IF ASC(Y$)=89 THEN 320
      GO TO 10

400   PRINT "OPTION - CALCULATE AND STORE TERM MARK"
      INPUT "CONFIRM OPTION (Y OR N) >";Y$
      IF ASC(Y$) <> 89 THEN 10
```

```
      REM      This function converts the term mark keeping only one
      REM      digit after the decimal point.
      DEF FNTR$(F$)
      D%=MATCH(".",F$,1)
      IF D% = 0 THEN\
         FNTR$=F$:RETURN
      F%=VAL(MID$(F$,D%+1,1))
      G%=VAL(MID$(F$,D%+2,1))
      IF G%>=5 THEN F%=F%+1
      NF$=LEFT$(F$,D%)+STR$(F%)
      FNTR$=NF$
      RETURN
      FEND

      INPUT "ENTER WEIGHT ON ASSIGNMENTS >";WA
      INPUT "ENTER WEIGHT ON FIRST MID-TERM >";W1
      INPUT "ENTER WEIGHT ON SECOND MID-TERM >";W2
      INPUT "ENTER WEIGHT ON FINAL EXAM >";W3
      PRINT: PRINT " TERM MARKS ARE: ":PRINT
      FOR I%=1 TO R2%(1)
         TTA=0.
         FOR K%=1 TO 10
            V$=FNR2$(I%,K%+3,1)
            IF V$="S" THEN TA=1. ELSE TA=0.
            TTA=TTA+TA
         NEXT K%
         MID1=VAL(FNR2$(I%,14,1))
         MID2=VAL(FNR2$(I%,15,1))
         FINAL=VAL(FNR2$(I%,16,1))
         TM=WA*TTA+W1*MID1+W2*MID2+W3*FINAL
         F$=STR$(TM): TM$=FNTR$(F$)
         PRINT FNR2$(I%,2,1);" _ ";TM$
         V$=FNR3$(I%,17,1,TM$)
      NEXT I%
      PRINT: PRINT "TERM MARKS HAVE BEEN ENTERED."
      GO TO 10

1001  REM Subroutine for entering marks or grades by section
      INPUT "ENTER SECTION NUMBER >";SN%
      IF SN%<1 OR SN%>4 THEN 1001
      PRINT
      I1%=0
      FOR I%=1 TO R2%(1)
         N%=VAL(FNR2$(I%,3,1))
         IF N%=SN% THEN\
            I1%=I1%+1:\
            PRINT FNR2$(I%,1,1);" _ ";FNR2$(I%,2,1);:\
            INPUT N$:\
            V$=FNR3$(I%,C%,1,N$):\
            T%(I1%)=I%
```

```
   N EXT I%
     RETURN

)01  REM Subroutine for changing any mark or grade.
     FOR K%=1 TO I1%
        PRINT
        PRINT FNR2$(T%(K%),1,1);" _ ";FNR2$(T%(K%),2,1);" _ ";
        PRINT FNR2$(T%(K%),C%,1)
        INPUT "  CHANGE MARK (Y OR N) >";Y$
        IF ASC(Y$)<>89 THEN 2100
        INPUT "  NEW MARK >";N$
        V$=FNR3$(T%(K%),C%,1,N$)
100  NEXT K%
     RETURN

2010 CLOSE 10,11,12
     END
```

BIBLIOGRAPHY

[1] Meldman, McLeod, Pellicore, Squire M. RISS: A Relational Data Base Management System For Minicomputers. Van Nostrand Reinhold Company, 1978.

[2] Date, C. J. Introduction to Data Base Management Systems. Addison-Wesley Publishing Company Inc., 1977.

[3] Martin, James. Computer Data-Base Organization. Prentice-Hall, Inc., 1977.

[4] Bohl, Marilyn. Information Processing. Science Research Associates Inc., 1976.

[5] Special Issue: Data Base Management Systems. Computing Surveys, Vol. 8, No. 1, Mar. 1976.

[6] OS Total Reference Manual. CINCOM Systems, Inc., Cincinnati, Ohio, 1978.

[7] Information Management System/Vertical Storage. Utilities Reference Manual. IBM Corporation, 1974.

[8] Astrahan, M. M. et al. System R - A Relational Approach to Database Management. ACM Trans. Data Base Systems, Vol. 1, No. 2, 1976.

[9] Data book 70, the EDP buyer's bible. Vol. 3, Datapro Research Corporation, N. J. 1980.

[10] CODASYL Systems Committee. Feature Analysis of Generalized Data Base Management Systems. New York: Association for Computing Machinery, May 1971.

[11] Codd, E. F. (ed.). Implementation of Relational Data Base Management Systems. Bulletin of ACM-SIGMOD, The Special Interest Group on Management of Data, Vol. 7, No. 3-4, 1975.

[12] Codd, E. F. A Relational Model of Data for Large Shared Data Banks. CACM Vol. 13, No. 6, June 1970.

[13] Whitney, V. K. M. RDMS - A Relational Data Management System. Proceedings Fourth International Symposium on Computer and Information Sciences, Miami. Plenum Press, 1972.

[14] Chamberlain, D. L. Relational Data Base Management System. Computing Surveys, Vol. 8, No. 1, March 1976.

[15] Astrahan, M. M. et al. System R - A Relational Approach to Database Management. ACM TODS 12, June 1976.

[16] Haseman and Whinston. Introduction to Data Management. Richard D. Irwin Inc., 1977.

[17] Ross, Ronald G. An Assessment of Current Data Base Trends. Data Base Monograph Series Q.E.D. Information Sciences.

[18] Palmer, Ian R. Data Base Systems: A Practical Reference. QED Information Sciences, Inc., 1975.

[19] Cohen Leo. Data Base Management Systems: A Practical Reference. Q.E.D. Information Sciences, Inc., 1976.

[20] Cardenas Alfonso. Data Base Management Systems. Allyn and Bacon, Inc. 1979.