(85)

AN INTERACTIVE MENU-DRIVEN VAX RELATIONAL DATABASE SYSTEM

728

Dissertation submitted in partial fulfilment of

the requirements for the Degree of

MASTER OF TECHNOLOGY IN COMPUTER SCIENCE

48p, bibliography

BY

MANOJ KUMAR GUPTA

JAWAHARLAL NEHRU UNIVERSITY LIBRARY

SCHOOL OF COMPUTER AND SYSTEM SCIENCES

JAWAHARLAL NEHRU UNIVERSITY

NEW DELHI

1989

CERTIFICATE

The research work embodied in this dissertation has been carried out at the School of Computer Sciences, Jawaharlal Nehru University, New Delhi-110007. This work is original and has not been submitted so far, in part or full, for any other degree or diploma of any University.

Manoj K. Gupta

(MANOJ KUMAR GUPTA)

Student

[illegible]

(Dr. P. C. SAXENA)

Superviser

[illegible]

(Prof. Karmeshu)

Dean

School of Computer Sciences,

Jawaharlal Nehru University,

New Delhi 110007

PREFACE

Of late database management has come under the more useful and powerful applications of computers. Many models of database have been proposed for effective and efficient implementation of data, for example, network, hierarchical, relational etc. Of these, the Ralational Data Model offers both the simplicity of design and efficiency at implementation.

In Jawaharlal Nehru University, we have the super mini-computer VAX-11/780, that supports both the VMS (version 4.4) and the Ultrix operating systems. The VMS also supports the software package on Relational Database System which we shall henceforth call as Rdb/VMS. Rdb/VMS(version 2.2), on which this project has been implemented, runs on both the VMS and the MicroVMS operating systems.

VAX Rdb/VMS is a relational database management system for VMS systems. It is a part of Vax Information Architecture. Rdb/VMS is intended to be used as a data access method by high-level language programs. Rdb/VMS includes RDO (relational database operator), an interactive utility for data definition, learning and prototyping.

The Rdb/VMS is currently useful only to knowledgeable database programmers with access to manuals and other facilities. The primary objective of this work is to bring this utility within the reach of layman by implementing a user-frienly interface. This interface is essentially a menudriven system. All the ruotines of the package are accessible from the menus. The user has to select an option alphabet to invoke a particular ruotine. The main menu drives the sub-menus.

A modular approach has been given to the design of the system. The major sub-systems are:

1.Data definition.

2.Data entry.

3.Data modification or editing.

4.Processing data.

The user can bypass to any sub-system depending upon his needs. All the menus tell the user how to proceed with necessary instructions.

The software has been implemented on VAX-11/780 mini-computer supported by Rdb/VMS(version 2.2) and operating system VMS(version 4.4). The programming language used for the implementation is VAX PASCAL.

Chapter 1 gives an introduction about relational data model and VAX/Rdb terminology. This chapter also describes in brief the approach and solution to the developed system.

Chapter 2 describes the methods of retrieving and updating data, RDO (relational database operater), RDML (relational data manipulation language), high-level language precompiled program, RDBPASCAL preprocessor interface and CALLABLE RDO interface (including the function RDB$INTERPRET). It also includes the main program algorithm and statement execution process. These explain the basic scheme of control logic of the menu-driver and also how input data from the user is synthesised into Rdb statements and passed to RDB$INTERPRET.

Chapter 3 describes defining database, fields and relations. It also explains VAX/Rdb datatypes and their conversion by the RDBPASCAL and RDB$INTERPRET interface. It gives an account of how to enter data values in the respective fields.

Chapter 4 is about editing the field values, that is how to insert, delete or modify them.

Chapter 5 describes the various statistical functions that can be used on the field values to give useful information.

ACKNOWLEDGEMENTS

I wish to express my gratitude to my superviser, Dr. P. C. Saxena, Associate Professor, School of Computer Sciences, Jawaharlal Nehru University, for his valuable guidance and co-operation. He was a constant source of encouragement thruoghout this project.

I would also thank Professor Karmeshu, Dean, School of Computer Sciences, Jawaharlal Nehru University, for his co-operation and for providing all the facilities in completing this work.

I am also thankful to my friends, Shri R. K. Gupta and Shri N. K. Jain for their help and useful discussions.

My thanks are also due to Shri Fateh Singh, Administrative Officer, Computer Center and Shri Randhir Singh for their co-operation and help in many ways.

[illegible]

CONTENTS

CHAPTER 1

INTRODUCTION

A relation is simply a two-dimensional table that has several properties. First, the entries in the table are single-valued; neither repeating groups nor arrays are allowed. Secondly, the entries in any column are all of the same kind. Further each column has a unique name and the order of the columns is immaterial.

Columns of a relation are referred to as attributes. Finally, no two rows in the table are identical and the order of the rows is insignificant. Each row of the relation is known as a tuple. If the relation has n columns than each row is referred to as n-tuple. Also, a relation that has n columns or n attributes is said to be of degree n. The following table shows the correspondences between different sets of terminology:

DATABASE ENTITY	RELATIONAL JARGON	VAX Rdb/VMS
Table	Relation	Relation
Column	Attribute	Field
Row	Tuple	Record

Every record stored in the database must have at least one field that can be used to locate a single record. This field is called a primary key. A primary key must have certain features that allow it to locate one record from all the records in the database. Two important characteristics of a primary key follow:

o Must not contain duplicate values.

o Must not contain null values.

This dissertation aims at providing the facilities of VAX/Rdb in a user-friendly environment. VAX/Rdb (Relational database package available on VAX-11/780 mini-computer) gives complete facilities for data definition, maintenance, and manipulation. The guidance to use the system is provided thruogh an on-line 'HELP' feature. It is very cumbersome for a novice to get started on the system using

"HELP" since it can not be used interactively. Moreover reading of lenthy manuals becomes necessary to implement any worthwhile application.

The present project brings the VAX/Rdb facility within easy reach of a layman by using an interactive menu-driven approach. The user is guided at every step thruogh a number of available options of which he can select anyone of his choice. In this way he can proceed in a predetermined manner, bypass some of the options or goback to where he had started.

The system has been developed in VAX PASCAL. All VAX languages that support the VAX Procedure Calling Standard can use the Callable RDO (Relational Data Operater) program interface like BASIC, C, FORTRAN, COBOL, PASCAL. PASCAL was chosen particularly because of more convenient file reading and writing, extensive use of which has has been made thruoghout the program. Moreover PASCAL supports the argument passing BY DESCRIPTOR mechanism, which is necessary to pass VAX Rdb/VMS statements from a PASCAL program to the Rdb interpreter. In any case the other languages could also be used.

A modular approach has been adopted in the desiging of the system to ensure easy readability, modification and debugging. The whole program is split in four major sub-modules i.e, data-definition, data-entry, data-modification and processing (to include some useful mathematical operations on the data).

The system incorporates the most basic and some of the more useful facilities of VAX/Rdb. The report lays down the designing approach and various algorithms needed for the sub-systems. The system can be easily enhanced later on as the need arises.

1.1 MENU-SYSTEM

Menu-system contains a nucleus called menu-driver, a program that takes control of the system on initiation and provides full interaction facilities to the user. It makes the approach modular. It loses control when an exit request is made.

The whole program has been partitioned into several modules:

* the format procedure
* data-entry procedure
* edit procedure

* data processing procedure

The MAIN PROGRAM consists of one big CASE statement. It displays the main menu and asks for an option. Upon reading the input, it transfers the control to that sub-system.

The FORMAT PROCEDURE contains the database defining module. It asks the name of database, the various fields, the relations alongwith the datatypes of the fields. Using this information it creates the database with requisite features.

The DATA-ENTRY PROCEDURE allows the user to store values in his database. For the convenience of the user it displays the name of the field and the relations along with their details if the user asks for them.

The EDIT PROCEDURE gives the user the facilities to modify, delete or insert values in the fields.

The PROCESS PROCEDURE includes the various useful operations that a user can perform on the fields of the database. They include sorting, mathematical functions like maximum, minimum, count, average, total etc.

CHAPTER 2

PROGRAM SCHEME DESCRIPTION

As a programming tool, VAX Rdb/VMS has several advantages:

- o The versatility of the data manipulation statements means that the database system itself can perform many of the tasks that would other wise be needed to be coded in a high level language.
- o The interactive environment, RDO, allows the user to prototype the applicationand its programs completely before writing the program.

Rdb/VMS provides several methods of retrieving and updating information in a database. They are:

2.1 RDO, THE INTERACTIVE RDB/VMS UTILITY.

RDO is the interactive utility for VAX Rdb/VMS. RDO lets the user type Rdb/VMS statements interactively and see the results immediately. RDO can be used for:

- o Defining and maintaining the database
- o Learning about Rdb/VMS
- o Testing and prototyping Rdb/VMS applications
- o Performing small-scale data manipulation operations

2.2 HIGH-LEVEL LANGUAGE PRECOMPILED PROGRAMS.

With minor adjustments the statements developed using RDO can be included in programs. Several precompilers are available providing support for VAX C, VAX COBOL, VAX BASIC, VAX FORTRAN, and VAX PASCAL programs containing embedded DML statements.

2.2.1 RDBPASCAL Preprocessor Interface

Using the RDBPASCAL preprocessor interface, the Rdb/VMS data manipulation statements can be directly included in the program. When the suorce program is precompiled, the preprocessor converts the DML statements to a series of equivalent PASCAL calls to Rdb/VMS. The program is then compiled as any other PASCAL program. At run time, Rdb/VMS executes the calls and returns any retreived data to the program.

2.3 RDML

The Relational Data Manipulation Language (RDML) is comprised of cluases, expressions and statements that can be embedded in C and PASCAL programs. These programs can be processed by the RDML preprocessor, which converts the RDML statements into a series of equivalent DIGITAL Standard Relatonal Interface (DSRI) calls to the database. Following a successful precompilation, the programmer can submit the resulting source code to the host language compiler.

2.4 CALLABLE RDO, HIGH-LEVEL CALL INTERFACE.

The Rdb/VMS language can be passed to the interpretive interface, Callable RDO, using simple calls from any language that adheres to the VAX Calling Standard.

While using the callable RDO program interface, the program communicates with Rdb/VMS using a callable procedure, RDB$INTERPRET. Unlike precompiler interfaces, the Callable RDO interface functions in an interpretive manner. When the program executes, the statements are passed to Rdb/VMS in the procedure calls to RDB$INTERPRET.

The interactive Rdb/VMS interface, RDO, then interprets and executes them.

Callable RDO is significantly slower than precompiled Rdb/VMS. Therefore it is used only when:

* An Rdb/VMS precompiler does not existfor the host language
* the program must perform data definition tasks

Though precompiled programs facilitate data manipulation, they can not be used for data-definition. Therefore Callable RDO interface must be used for data-definition tasks.

2.5 **RDB$INTERPRET**

RDB$INTERPRET is declared as an external integer (longword) function. In the calling sequence both Rdb/VMS statements and host variables are passed. The call to RDB$INTERPRET returns a status value describing the success or failure of the statement. The return status value is a system-wide condition value that is either:

* Success
* A unique Rdb/VMS symbolic error code

RDB$INTERPRET requires all parameters (the Rdb/VMS statement and host variables) to be passed 'BY DESCRIPTOR'. The PASCAL format of the RDB$INTERPRET calling sequence is:

o **ret-status = RDB$INTERPRET(%STDESCR 'rdb statement'[,[%STDSCR][%DESCR] host-var....]);**

Arguments

* **ret-status :** A program variable that holds the longword integer describing the success or failue of the call. The program tests the value of ret-stat and optionally branches to an exception condition handling routine.

* **rdb-statement :**The Rdb/VMS statement being passed to Rdb/VMS.

* **host-var :** A host-variable passed to Rdb/VMS as part of the data manipulation statement.

The rdb statement can be included in the calling sequence directly as a string literal. However, the length of some Rdb/VMS statement may produce unwieldy code in the call to RDB$INTERPRET. Instead the Rdb/VMS statement string literal is assigned to a string variable. Then string variable is passed in the calling sequence.

The main program consists of one big Case staement. It displays the opening menu and asks for the options. Depending upon tne input it calls the appropriate procedure.

The scheme of logic has been taken according to the facilities available in the PASCAL language. The system is user-friendly and also takes input from the user. Besides the syntax of the Rdb statement shuold not be visible to user. The syntax as well as the input(like the names of the fields and relations) have to be put together to make a valid statement, which can then be passed to RDB$INTERPRET to be executed.

Therefore for convenience a separate file is opened for writing the input as well as the syntax. As soon as the user wants a particular utility, the syntactical statement pertaining to it are written on that file. The user-supplied input is also put in its proper place by the program. The contents of the file are then read into a packed array. And finally the contents of this array are passed to the function RDB$INTERPRET which executes them. The contents of the file are erased before the next statement is writen.

A menu-driven system must also provide facilities for going back to the system or to the main opening menu whenever user wants. To go back to the system the user has to press CTRL/Y. To go back to opening menu the user should press '!'. Before taking any input from the user the program checks for '!' and if it is encountered then the control is transferred to the opening menu.

Following is the algorithm for the opening menu:

ALGORITHM

```
1.  Display opening menu.

2.  Get select code(=SC).

3.  If SC=CTRL/Y, exit to system

4.  If SC=1, invoke format

5.  If SC=2, invoke enter

6.  If SC=3, invoke edit

7.  If SC=4, invoke process

8.  If SC=8, invoke show
```

9. If SC = ANY OTHER KEY, display error message

10. Go to step 2

The various data-manipulation operations are done through transactions.

2.6 TRANSACTIONS

Rdb/VMS allows many users to access a database at the same time. To avoid conflicts and data inconsistencies Rdb/VMS requires each user to identify a database activity, called a transaction. A transaction is an operation on the database that must complete as a unit or not complete at all. They end either by a COMMIT or ROLLBACK statement.

Using the COMMIT statement makes the changes parmanent. ROLLBACK statement is used to undo the changes made to the database within the scope of a transaction.

The following Rdb/VMS data manipulation statements can be typed at the terminal in an RDO session or included in high level programs. The folowing table introduces the data manipulation statements:

Statement	Description
Commit	Ends a transaction, makes changes parmanent
Database	Declares a database
Erase	Erases one or more records from an existing relation
Fetch	Advances pointer for a record stream to next record
Finish	Detaches process from database
For	Executes a statement,once for each record in a record stream
Get	Retrieves field values from a record stream and assigns them to variables in a program
Invoke	Declares a database, either to RDO or in a host language program
Modify	Modifies one record in an existing relation
On-error	Specifies statements to be performed if an error occurs during execution of a data manipulation statement
Rollback	Undoes the changes made during transaction
Start_Stream	Declares and opens a record stream
Start_ Transaction	Initiates a transaction, which is a series of data manipulation statements executed as a unit
Store	Stores one record in an existing relation

CHAPTER 3

DATA DEFINITION AND ENTRY

Rdb/VMS supports eight existing VAX data types, and a special Rdb/VMS data type. the existing data types are:

* Signed WORD
* Signed LONGWORD
* Signed QUADWORD
* F_FLOATING
* G_FLOATING
* DATE
* TEXT
* VARYING STRING

The special data type is: SEGMENTED STRING

3.1 DATA TYPE CONVERSIONS

The host language program(VAX PASCAL) may access an Rdb/VMS data type that is not supported by the host language. in these cases, Rdb/VMS performs data type conversions wherever possible before passing database values to host variables. Rdb/VMS converts data types for:

* precompiled programs
* callable RDO programs

3.1.1 PRECOMPILED PROGRAM DATA TYPE CONVERSIONS

Precompilers declare a set of variables that act as an intermediate between host variables and database values. the data types assigned to these intermediate variables depend on the data type of the database field being accessed and the precompiler being used.

When a host language data type is the same as the data type of the database value, the precompiler declares a variable of that data type and no data type conversion takes place. However, when host language does not support Rdb/VMS data type, the precompiler declares an intermediate variable that is supported by host language.

3.1.2 CALLABLE RDO PROGRAM DATA TYPE CONVERSION

For Callable RDO programs host variable data types are selected that are compatible with the database data types being accessed. When the host language does not support the Rdb/VMS data type, Rdb/VMS performs the data type conversions as listed in the table.

RDBPASCAL-Generated Data Typees for VAX PASCAL

Rdb/VMS Datatype	VAX PASCAL Datatype
SIGNED WORD	[WORD]-32768..32767
SIGNED LONGWORD	INTEGER
SIGNED QUADWORD	[BYTE(8)]RECORD END
F_FLOATING	REAL
G_FLOATING	DOUBLE
DATE	[BYTE(8)]RECORD END
TEXTn	PACKED ARRAY [1..n] OF CHAR

The following steps show the algorithm for the FORMAT sub-menu

ALGORITHM

1. Display format menu
2. Get database name
3. Define database
4. Get the no. of fields (henceforth referred as nof)
5. While nof < maximum no. of fields allowed, goto step 6
6. Get the name and datatype of each field and craete them

7. Get the definition of the relations and create them
8. Save the database thus created
9. Exit to opening menu

3.2 DEFINING DATABASE

The beginning is made by naming a database, which shall contain all the fields and the relations. This database will have to be invoked each time any operation is to be done. The RDO command for defining database is "DEFINE DATABASE 'NAME'.". Following are the main steps:

1. Open file1 and prepare it for writing
2. Write "DEFINE DATABASE" into file1
3. Read the database name from the user (keyboard) and write it onto file1
4. If "!" is encountered goto opening menu
5. Else continue writing to file1 until eoln is read
6. Read fieldname from keyboard and write it ao file1 until eoln

7. Read contents of filel into packed array c

8. Pass c as argument to RDB$INTERPRET

9. Write blanks in c

3.3 DEFINING FIELDS

Next the fields are defined. The RDO statement is "DEFINE FIELD 'NAME' DATATYPE IS 'NAME'".

1. Read no. of fields (nof) user wishes to use

2. Display the datatypes fields can have

3. For i = 1 to nof do

4. Rewrite filel

5. Write "DEFINE FIELD" to filel

6. Read fieldname from keyboard and write it ao filel until eoln

7. Write "DATATYPE IS" to filel

8. Read datatype from keyboard and write it ao filel until eoln

9. Read contents of filel into packed array c

10. Pass c as argument to RDB$INTERPRET

11. Write blanks in c

3.4 DEFINING RELATION

And finally the relation name is defined to which the fields belong. The rdo statement is "DEFINE RELATION 'NAME' [FIELDNAME 1],[FIELDNAME 2],..."

1. Rewrite filel

2. Write 'DEFINE RELATION' into filel

3. Read the RELATION name from the user (keyboard) and write it onto filel

4. Read fieldnames from keyboard and write it ao filel until eoln

5. Read contents of filel into packed array c

6. Pass c as argument to RDB$INTERPRET

7. Write blanks in c

Some of the important RDO terms and definitions for read-write in the database are described below:

3.5 COMMANDS AND DEFINITIONS

1. **STARTTRANSACTION:**

STARTTRANSACTION initiates a group of statements that Rdb executes as a unit. All the statements that modify records within a transaction take effect when the transaction is completed, or none of them do. If the transaction is ended with the COMMIT statement, all the statements within the transaction execute. If the transaction is ended with a ROLLBACK statement, none of the statements take effect.

JAWAHARLAL NEHRU UNIVERSITY

TH-2949

2. **CONTEXTVARIABLE:**

A temporary name that identifies a relation in a record stream to Rdb. Once we have associated a context variable with a relation, we use the context variable to refer to fields from that relation. In this way, Rdb always knows which field from which relation being referred to.

DISS
681 302
9 959
in

A context variable must be used in every data manipulation statement and in every data definition statement that uses a record selection expression.

If several record streams are being accessed at once, the context variable allows to distinguish between fields from different record streams, even if different fields have the same name.

If several record streams are being accessed at once that consist of the same relation and fields within that relation, context variables allow to distinguish between the two record streams.

3. FOR Statement:

The FOR statement executes a statement or group of statements once for each record in a record stream formed by a record selection expression. FOR statements can be nested within other FOR statements to establish relationships for outer joins.

The program can use either FOR statements or STARTSTREAM statements to establish record streams. Both methods can be used in one program. However, the FETCH statement can not be used to advance the pointer in a record stream established by a FOR statement. The

FOR statement automatically advances to the next record.

4. **STARTSTREAM:**

Declares and opens a record stream. The STARTSTREAM statement:

- o Forms a record stream from one or more relations. The record selection expression determines the records in the record stream.

- o Places a pointer for that stream just before the first record in this stream.

The FETCH statement must be used to advance the pointer one record at a time through the stream and other RDML statements (for example, MODIFY and ERASE) to manipulate each record.

5. **FETCH:**

Retrieves the next record from a record stream. The FETCH statement is used:

- o After a STARTSTREAM statement

- o Before any other RDML statements that affect the context established by the STARTSTREAM statement

The FETCH statement advances the pointer for a record stream to the next record of a relation. Unlike the FOR statement, which advances to the next record automatically, the FETCH statement allows us the explicit control of the record stream. For instance, one might use the FETCH statement to print a report where the first six rows have five columns, and the seventh row only three

6. **COMMIT:**

Ends a transaction and makes permanent any changes you made during that transaction to the database.

7. **ROLLBACK :**

Terminates a transaction and undoes all changes made to the database since the program's most recent STARTTRANSACTION statement or since the start of the specified transaction.

8. **ONERROR**

The ON ERROR clause specifies the statement(s) the host language performs if an error occurs during the execution of the associated RDML statement.

The ON ERROR clause can be used in all RDML statements except the DATABASE statement.

3.6 DATA ENTRY

The following RDO statement is used to assign values to the fields:

o **STORE Statement**

Inserts a record into an existing relation. We can add a record to only one relation with a single STORE statement. The statements between the keywords STORE and ENDSTORE form a context block. The RDO format for the store staement is: ` STORE C IN REL USING C.FIELD = "VALUE" ENDSTORE '

Algorithm for entering data follows:

ALGORITHM

1. Display menu

2. Read the database name

3. Invoke the database

4. Rewrite filel

5. write 'STORE C IN ' to filel

6. read the relation name and write it to filel

7. Write 'USING C." to filel

8. Read the name of the field to which value is to be asssigndand write it to filel

9. Write ' = '' to filel

10. read the value to be stored and write it to filel

11. Write "' ; ENDSTORE ' to filel

12. Read contents of filel into packed array c

13. Pass c as argument to RDB$INTERPRET

14. Write blanks in c

15. COMMIT or ROLLBACK as the user requires

16. Exit to the opening menu

CHAPTER 4

DATA-EDITING

Edit gives the facility to modify, erase, or insert a new record. There is also a facility to display the details of the database, fields and relations in case the user has forgotten any of these.

Following is the scheme for EDIT sub-menu:

ALGORITHM

```
1.  Display edit menu

2.  Get select code

3.  If SC = !, exit to opening menu

4.  If SC = 1, invoke insert

5.  If SC = 3, invoke delete
```

6. If SC = 2, invoke modify

7. If SC = 8, invoke show

8. If SC = CTRL/Y goto the system

9. If SC = any other key, display opening menu

4.1 COMMANDS AND DEFINITIONS

1. **MODIFY:**

Changes the value in a field or fields in one or more records from a relation or open stream

Before using a MODIFY statement:

- o A READWRITE transaction must be started
- o A record stream with a FOR statement or STARTSTREAM statement must be established

The context variables referenced in a MODIFY statement must be the same as those defined in the FOR or STARTSTREAM statement.

The RDO format for the MODIFY statement is: ` FOR H IN REL MODIFY H USING H:FIELD = 'VALUE' ENDFOR '

Detailed algorithm for MODIFY sub-section:

1. Read the database name
2. Invoke the database
3. Rewrite filel
4. Write 'FOR M IN ' to filel
5. Read the relation name and write it to filel
6. Write 'MODIFY M USING M.' to filel
7. Read the name of the field to which value is to be asssignd and write it to filel
8. Write ' = '' to filel
9. read the value to be modified and write it to filel
10. Write ' ' ENDFOR ' to filel
11. Write COMMIT or ROLLBACK as the user requires

12. Read contents of file1 into packed array c

13. Pass c as argument to RDB$INTERPRET

14. Write blanks in c

15. Exit to the opening menu

2. ERASE

Deletes records from a relation or open stream, one at a time. The RDO format for the ERASE statement is:
` FOR E IN REL WITH E.FIELD = "VALUE" ERASE E ENDFOR '

Main steps in ERASE sub-section are

1. Read the database name

2. Invoke the database

3. Rewrite file1

4. Write "FOR E IN " to file1

5. Read the relation name and write it to file1

6. Write "WITH E." to file1

7. Read the name of the field whose value is to be erased and write it to filel

8. Write " = '" to filel

9. Read the value to be erased and write it to filel

10. Write " ' ERASE E ENDFOR " to filel

11. Write COMMIT or ROLLBACK as the user requires

12. Read contents of filel into packed array c

13. Pass c as argument to RDB$INTERPRET

14. Write blanks in c

15. Exit to the opening menu

3. INSERT

STORE statement inserts a record into an existing relation. We can add a record to only one relation with a single STORE statement. The statements between the keywords STORE and ENDSTORE form a context block.

The RDO format for the store staement is: ` STORE C IN REL USING C.FIELD = "VALUE" ENDSTORE '

The algorithm for INSERT is similar to that used of STORE in the previous chapter.

CHAPTER 5

PROCESSING DATA

For processing data VAX/Rdb provides several statistical fuctions like max, min, count, total and average. Besides these SORTED BY facility provided by Rdb has also been included in this module.

5.1 STATISTICAL FUNCTIONS

Calculate values based on a value expression for every record in a record stream. A value expression is not specified for the COUNT statistical function because it operates on the record stream formed by the RSE, not on a value expression. When using the AVERAGE, MAX, MIN, and TOTAL statistical functions, we specify a value expression and a record selection expression (RSE). Rdb then:

- o Evaluates the value expression for each record in the record stream formed by the RSE
- o Calculates a single value based on the results of the first step

The RDML Statistical functions are:

- o AVERAGE
- o COUNT
- o MIN
- o MAX
- o TOTAL

The folowing table shows statistical expressions and their result:

Statistical Expression	Result
AVERAGE	Average of non-missing field values in current stream
COUNT	Number of records in current stream
MAX	Largest value of field in current stream
MIN	Smallest value of field in current stream
TOTAL	Sum of values of field in current stream

The algorithm for PROCESS sub-menu is:

ALGORITHM

```
 1.  Display process menu

 2.  Get the database name

 3.  Invoke the database

 4.  Read the option

 5.  If SC = 1, find maximum

 6.  If SC = 2, find minimum

 7.  If SC = 3, count

 8.  If SC = 4, find average

 9.  If SC = 5, total

10.  If SC = 6, sort

11.  If SC = 8, show

12.  If SC = !, display opening menu

13.  If SC = CTRL/Y , goto system
```

14. If SC = any other key display opening menu

15. Display opening menu

The description of various statistical and associated RDO statements occurs below:

5.2 COMMANDS AND DEFINITIONS

1. GET:

The GET statement is used to retrieve one, several, or all the fields in a database record. The statitical GET statement is used to retreive statitical values from the database. The GET statement is a read operation. The result of the statistical expreesion can be retreived directly without processing each record in the record stream. The result of a GET statement is always numeric. So the by-descriptor mechanism must be used for any field that receives the result of a statistical GET statement.

The Rdb format is: ` GET HOST-VAR = COUNT OF R IN RELATION WITH R.FIELD = 'VALUE' ENDGET '

2. **COUNT:**

Returns the number of records in a record stream specified by a record selection expression. The COUNT function differs from other statistical functions because it operates on the record stream defined by the record selection expression rather than on the values in that record stream.

The Rdb format is: ` PRINT COUNT OF R IN REL '. The steps are:

1. Read the database name from the keyboard and invoke it
2. Rewrite file1
3. Write 'PRINT COUNT OF C IN' to file1
4. Read the relation name and write it to file1
5. Read contents of file1 into packed array c
6. Pass c as argument to RDB$INTERPRET
7. Write blanks in c

8. Exit to the opening menu

3. **SORTED:**

Sorts the records in the record stream by the values of specific fields. Sorting is done on a database field value expression, called a sort key. The sort key determines the order in which Rdb returns the records in the record stream. The default sorting order is ascending order.

The Rdb format is: ` FOR S IN REL SORTED BY "ORDER" S.SORT-FIELD PRINT S.FIELD ENDFOR '.Steps are:

1. Read the database name

2. Invoke the database

3. Rewrite filel

4. Write 'FOR S IN ' to filel

5. Read the relation name and write it to filel

6. Write 'SORTED BY' to filel

7. Read 'ascending' or 'descending' from the user and write to filel

8. Write 'S. ' to filel

9. Read the name of the SORTFIELD to which value is to be asssignd and write it to filel

10. Write 'PRINT S. '' to filel

11. Read the name of the field whose sorted values are required and write it to filel

12. Write ' ENDFOR ' to filel

13. Read contents of filel into packed array c

14. Pass c as argument to RDB$INTERPRET

15. Write blanks in c

16. Exit to the opening menu

4. **MAX:**

Returns the highest value for a value expression for all records specified by a record selection expression.

The Rdb format is: ` PRINT MAX M.FIELD OF M IN REL '. Steps are:

1. Read the database name from the keyboard and invoke it

2. Rewrite filel

3. Write "PRINT MAX M." to filel

4. Read the FIELD name and write it to filel

5. Write "OF M IN" to filel

6. Read the relation name, write it to filel

7. Read contents of filel into packed array c

8. Pass c as argument to RDB$INTERPRET

9. Write blanks in c

10. Exit to the opening menu

5. **MIN**

Returns the lowest value for a value expression for all records specified by a record selection expression.

The Rdb format is: ` PRINT MIN M.FIELD OF M IN REL '. Steps are:

1. Read the database name from the keyboard and invoke it

2. Rewrite filel

3. Write 'PRINT MIN M.' to filel

4. Read the FIELD name and write it to filel

5. Write 'OF M IN' TO FILE1

6. Read the relation name, write it to filel

7. Read contents of filel into packed array c

8. Pass c as argument to RDB$INTERPRET

9. Write blanks in c

10. Exit to the opening menu

6. **TOTAL:**

Returns the sum of the values specified by a record selection expression. The value expression must be a numeric data type.

The Rdb format is: ` PRINT TOTAL T.FIELD OF T IN REL '.Steps are:

1. Read the database name from the keyboard and invoke it
2. Rewrite file1
3. Write 'PRINT TOTAL T.' to file1
4. Read the field name and write it to file1
5. Write 'OF T IN' TO FILE1
6. Read the relation name, write it to file1
7. Read contents of file1 into packed array c
8. Pass c as argument to RDB$INTERPRET
9. Write blanks in c
10. Exit to the opening menu

7. AVERAGE:

Determines the arithmetic mean of values for all records specified by a record selection expression.

The Rdb format is: ` PRINT AVERAGE A.FIELD OF A IN REL '. Steps are:

1. Read the database name from the keyboard and invoke it

2. Rewrite file1

3. Write "PRINT AVERAGE A." to file1

4. Read the field name and write it to file1

5. Write "OF A IN" TO FILE1

6. Read the relation name, write it to file1

7. Read contents of file1 into packed array c

8. Pass c as argument to RDB$INTERPRET

9. Write blanks in c

10. Exit to the opening menu

CHAPTER 6

CONCLUSIONS AND RECOMMENDATIONS

VAX/Rdb is one of the most comprehensive relational database available. It not only provides facilities for data-definition, storage and handling but also for proper privilege protection and error-message handling. The complete power of the system can be realised only when it is used either thruogh RDO or a precompiled high-level language program.

Due to the needs of a menu-driven system for use of a layman, CALLABLE RDO interface must be used. Since CALLABLE RDO is considerably slow, some power of the VAX/Rdb has to be sacrificed in order to have a reasonable speed. Accordingly error checking through RDB$SIGNAL and SYS$PUTMSG have not been incorporated. Also since the system is supposed to be used by novices, they have not been asked to define constraints in the definitions. Instead the program itself defines the constraints and checks if they are violated. For the same reasons the setting of privilege protection or accesses by the user has been avoided.

The use of the system is restricted by the contents of the existing menus. To add any operation, a program would have to be written and properly linked to the system which is a tedious job for an ordinary user. Also, after having once become familiar with the system, the user would find the interaction unnecessary.

The transfer control could have been done using system ruotines providing masking of characters. Instead the input from the keybaord is scanned and if "!" is encountered, control is transferred to the main program. Using this module in VAX C could be done because it provides a run-time-library function for scanning the keyboard.

While editing the record streams the user can edit only one record at a time, whereas in RDO he could form a record stream, and make changes in records selected by a record-selection expression. This limitation has to be put because it can not be known in advance what RSE format user might choose.

While defining fields the size is not specified by the user. The sizes for various datatypes have been declared fixed by the program. A suitable message is flashed to the user if he exceeds these limits.

Some improvements can be effected with some more programming:

1. The option to change the field and relation definitions could be provided.

2. Instead of using "SHOW ALL"statement, which displays all the information, the " SHOW FIELDS" and "SHOW RELATIONS" statements can be used to display selective information.

3. Maskable characters can be used to transfer control instead of periodically scanning the input.

4. The option to define constraints for fields can be incorporated.

5. The access rights definition can be included so that while a few can both read and write to the database, others can only read. Presently, the only diffrentiation between the users is according to the names of their datatbase.

6. The RDO HELP feature which is otherwise available to users can also be included for better customization.

7. A choice could be given to user between the interactive and the non-interactive modes so that he could bypass the interaction once he has become familiar with the system

8. System services can be used to maintain backup files of the running program so that in case power fails during execution, the user does not have to start all over again.

This project has been an attempt at providing a reasonably efficient menu-driven database system to new users of the computer. Hence care was takan to make it as user-friendly as possible. In the attempt some sophistication of the VAX/Rdb had to be sacrificed. Nevertheless the report lays down the fundamental program and algorithm structure upon which suitable additions can be made to make the system more useful.

BIBLOGRAPHY

1. Kroenky, D.M. "Database Processing: Fundamentals Design, Implementation" Galgotia Publications (1986).

2. "VAX Rdb/VMS Guide to Data Manipulation" Digital Equipment Corporation (1985).

3. "VAX Rdb/VMS Guide to Database Design and Definition" Digital Equipment Corporation (1985).

4. "VAX Rdb/VMS Guide to Database Administration and Maintenance" Digital Equipment Corporation (1985).

5. "VAX Rdb/VMS Reference Manual" Digital Equipment Corporation (1987).

6. "VAX Rdb/VMS Guide to Programming" Digital Equipment Corporation (1987).

7. "RDML Reference Manual" Digital Equipment Corporation (1987).

8. Date, C.J. "An Introduction to Database system" Addison-Wesley (1974).

9. "VAX VMS(4.4) User Manual" Digital Equipment Corporation (1985).

APPENDIX A

Sample run for the FORMAT sub-system

```
                                MAIN MENU
Press the option number according to your choice:
1....To define the database
2....To enter the data
3....To edit
4....To process data
8....To show details of existing fields and relations
CTRL/Y....To go back to the system
Now enter your option
INPUT---1

Press
      1________to format');
      !________to opening menu');
      CTRL/Y________to system');
INPUT---1

                        FORMAT
Format allows you to name your database, and define fields.
Press RET key after entering any name.
First of all, type the name of database

INPUT---MYDATA

Now enter the no. of fields you wish to use

INPUT---2

Now you are ready to enter the description of the fields,
namely the name and the datatype.
The datatype can be anyone of the following:
      * Signed longword (integer)');
      * F_floating (real)');
      * date');
      * text '');

Fieldname
INPUT---NAME
Fieldtype
INPUT---TEXT

Fieldname
INPUT---CLASS
```

```
Fieldtype
INPUT---SIGNED LONGWORD

 Enter the name of the relation
INPUT---PERS

If you wish to make the transaction parmanent
type 1, otherwise type 0
INPUT---1

                                   MAIN MENU
Press the option number according to your choice:
1....To define the database
2....To enter the data
3....To edit
4....To process data
8....To show details of existing fields and relations
CTRL/Y....To go back to the system
```

APPENDIX B

```
 (* PROGRAM FOR MENU-DRIVEN VAX/RDB *)

program p (input,output);

label 100;
type
c = packed array[1..100] of char;
(* common host var for single line statements *)

a= array[1..100] of char;

var     nof,n,m,i :integer;
      file1 : FILE OF char;
      c1,c2,c3,c4:c;
      a1:a;
      ch,op:char;

function RDB$INTERPRET
(%STDESCR X:packed array [1..100] of char ) : integer;
external;
function RDB$SIGNAL : INTEGER;
EXTERNAL;

procedure ho(var b:c) ;
(* passes argument to RDB$INTERPRET *)
 begin
 i := RDB$INTERPRET(%STDESCR b);
 end;

provedure moveblank(var bl:c);
(* puts blank spaces in the array  *)

begin
  for i := 1 to 100 do begin
    bl[i] := ' ';
    end
end;

procedure invoke;     (*      invokes the database    *)

begin
rewrite (file1);
write(file1,'I','N','V','O','K','E',' ');
write(file1,'D','A','T','A','B','A','S','E');
WRITE(file1,' ','F','I','L','E','N','A','M','E',' ','"');
```

```
writeln ('The DATABASE name you wish to work on_____');
      repeat
      read (ch);
if (ch <> '!') then begin
      write(filel)    end
else goto 100;
        until eoln;
      READLN;
WRITE(filel,'*');

reset (filel);
while ( not eof(filel)) do begin
read (filel, cl[i]);
i := i+1;     end;

i :=1;
ho(cl);
moveblank(cl);
end;          (* of invoke  *)

procedure show;
(* shows details of the fields & relations *)

begin
rewrite (filel);
invoke;
write(filel,'S','H','O','W',' ',' ',' ','A','L','L');

reset (filel);
while ( not eof(filel)) do begin
read (filel, cl[i]);
i := i+1;     end;

i :=1;
ho(cl);
moveblank(cl);
END;          (* of show *)

procedure mainmenu;
(*   displays main menu   *)

begin
writeln ('                           MAIN MENU');
writeln ('Press the option number according to your
choice:');
writeln ('1....To define the database');
writeln ('2....To enter the data');
writeln ('3....To edit');
writeln ('4....To process data');
writeln('8....Show details of existing fields and
relations);
```

```
writeln ('CTRL/Y....To go back to the system');
end;

procedure format;
(* defines database, fields & relations  *)

(* of format *)
begin
writeln('Press');
writeln ('    1________to format');
writeln ('    !________to opening menu');
writeln ('    CTRL/Y________to system');

read(op);
case  op of
'1':begin
writeln ('                      FORMAT ');
writeln ('Format allows you to name your database, ');
write('and define fields');
write ('Press RET key after entering any name. ');
writeln ('First of all, type the name of database____');

rewrite(file1);
write (file1,'D','E','F','I','N','E',' ');
write (file1,'D','A','T','A','B','A','S','E',' ');
         repeat
         read (ch);
if (ch <> '!') then begin
         write(file1)
                  end
else goto 100;
           until (eoln)   ;
         readln;
write (file1,' ','.');
reset (file1);

while ( not eof(file1)) do begin
read (file1, c1[i]);
i := i+1;      end;

i :=1;
ho(c1);
moveblank(c1);
writeln ('Now enter the no. of fields you wish to use__');
read(nof);

writeln ('Now you are ready to enter the description ');
write('of the fields, namely the name and the datatype.');
write(' The datatype can be anyone of the following:');
writeln ('    * Signd longword (integer)');
writeln ('    * F_floating (real)');
writeln ('    * date');
writeln ('    * text ');

for n := 1 to nof do begin

writeln ('Fieldname_________');
rewrite(filel);
write (filel,'D','E','F','I','N','E',' ');
write(filel,'F','E','I','L','D',' ');
      repeat
      read (ch);
if (ch <> '!') then begin
      write(filel)    end
else goto 100;
        until eoln;
      READLN;
write(filel,'D','A','T','A','T','Y','P','E',' ','I','S','
');
writeln ('Fieldtype_____________');
      repeat
      read (ch);
if (ch <> '!') then begin
      write(filel)    end
else goto 100;
        until eoln;
      readln;
write (filel,' ','.');
reset (filel);

while ( not eof(filel)) do begin
read (filel, cl[i]);
i := i+1;     end;
i :=1;
ho(cl);
moveblank(cl);

end;   (* of defining fields *)

writeln (' Enter the name of the relation______');
rewrite(filel);
write (filel,'D','E','F','I','N','E',' ');
write (filel,'R','E','L','A','T','I','O','N',' ');
      repeat
      read (ch);
if (ch <> '!') then begin
      write(filel)    end
else goto 100;
        until eoln;
      READLN;
write (filel,' ','.');
reset (filel);
```

```
while ( not eof(filel)) do begin
read (filel, cl[i]);
i := i+1;     end;

i :=1;
ho(cl);
moveblank(cl);

       '!' : goto 100;

 otherwise
begin    writeln ('Please enter a valid option');
        goto 100;
end;
end;  (* of case *)
end;   (* of format  *)

procedure store;
(* stores the value for a field  *)

begin
writeln('Press');
writeln ('1__to store');
writeln ('8__show details of fields and relations);
writeln ('!__to opening menu');
writeln ('CTRL/Y________to system');
read(OP);

case  op of
       '1':    begin
writeln ('ENTER lets you enter the values to fields.
write('Press RET key after entering any name.To
begin,first');
write ('type the name of relation _________');
invoke;
ho(c4);
rewrite(filel);
write(filel,'S','T','O','R','E',' ','C',' ','I','N',' ');
       repeat
       read (ch);
if (ch <> '!') then begin
       write(filel)    end
else goto 100;         until eoln;
       READLN;
write (filel,'U','S','I','N','G',' ','C','.');
writeln ('The field name______________');
       repeat
       read (ch);
if (ch <> '!') then begin
       write(filel)    end
else goto 100;
```

```
          until eoln;
        READLN;
write (filel,' ','=',' ','*');
writeln ('The field value_____________');
        repeat
        read (ch);
if (ch <> '!') then begin
        write(filel)     end
else goto 100;
          until eoln;
        READLN;
write (filel,'*',';',' ');
write (filel,'E','N','D','_','S','T','O','R','E',' ');
writeln ('If you wish to make the transaction parmanent ');
write('type 1, otherwise type 0______');
read (m);
case m of
        1 : ho(c2);
        0 : ho(c3);
END;

reset (filel);
while ( not eof(filel)) do begin
read (filel, cl[i]);
i := i+1;     end;

i :=1;
ho(cl);
moveblank(cl);
goto 100;
end;   (* of casel *)

'!' : goto 100;
'8' : show;

 otherwise
begin    writeln ('Please enter a valid option');
       goto 100;
end;
end;  (* of case *)
end;                     (* of enter *)

procedure edt;
(* inserts, deletes & modifies field values *)
begin

writeln ('EDT gives you the facility for modifying ');
write('or erasing a particular');
write (' field value. The varios options available are ');
writeln ('    1_____insert');
writeln ('    2_____modify');
writeln ('    3______erase');
writeln('     8______show details of fields and relations');
writeln ('    !______to opening menu');
writeln ('    CTRL/Y________to system');
read(op);

case op of
      '2': BEGIN
writeln('MODIFY lets you change the entry in a field
value.);
write ('Press RET key after entering any name. ');
write('To begin first type the name of relation ________');
invoke;
ho(c4);
rewrite(filel);
write (filel,'f','O','R',' ','H',' ','I','N',' ');
      repeat
      read (ch);
if (ch <> '!') then begin
      write(filel)    end
else goto 100;
until eoln;
      READLN;
write (filel,'M','O','D','I','F','Y',' ','H',' ');
write(filel,'U','S','I','N','G',' ','H','.');
writeln ('The field name_____________');
      repeat
      read (ch);
if (ch <> '!') then begin
      write(filel)    end
else goto 100;
until eoln;
      READLN;
write (filel,' ','=',' ','"');
writeln ('The field value_____________');
      repeat
      read (ch);
if (ch <> '!') then begin
      write(filel)    end
else goto 100;
until eoln;
      READLN;
write (filel,'"',' ');
write (filel,'E','N','D','_','F','O','R',' ');
writeln (' If you wish to make the transaction ');
write('parmanent type 1, otherwise type 0______');
read (m);
case m of
      1 : ho(c2);
      0 : ho(c3);
END;
```

```
reset (file1);
while ( not eof(file1)) do begin
read (file1, c1[i]);
i := i+1;      end;

i :=1;
ho(c1);
moveblank(c1);
goto 100;
end;                                (* of modify *)

'3' : begin
writeln;
write('ERASE lets you delete the entry in a field value.');
write ('Press RET key after entering any name. To begin ');
write('first type the name of relation ____________');
invoke;
ho(c4);
rewrite(file1);
write (file1,'f','O','R',' ','E',' ','I','N',' ');
      repeat
      read (ch);
if.(ch <> '!') then begin
      write(file1)    end
else goto 100;
   until eoln;
      READLN;
write (file1,'w','i','t','h',' ','e','.')
writeln ('The field name____________');
      repeat
      read (ch);
if (ch <> '!') then begin
      write(file1)    end
else goto 100;
until eoln;
      READLN;
write (file1,' ','=',' ','"');
writeln ('The field value____________');
      repeat
      read (ch);
if (ch <> '!') then begin
      write(file1)    end
else goto 100;
until eoln;
      READLN;
write (file1,'"',' ');
write (file1,'E','R','A','S','E',' ','E',' ');
write (file1,'E','N','D','_','F','O','R',' ');

writeln (' If you wish to make the transaction ');
write('parmanent type 1, otherwise type 0______');
```

```
read (m);
case m of
      1 : ho(c2);
      0 : ho(c3);
END;

reset (file1);
while ( not eof(file1)) do begin
read (file1, c1[i]);
i := i+1;     end;
i :=1;
ho(c1);
moveblank(c1);
goto 100;
end;

'1':
begin
writeln ('ENTER lets you enter the values to fields.);
write ('Press RET key after entering any name. To ');
write('begin, first type the name of relation ______');
invoke;
ho(c4);
rewrite(file1);
write (file1,'S','T','O','R','E',' ','C',' ','I','N',' ');
      repeat
      read (ch);
if (ch <> '!') then begin
      write(file1)    end
else goto 100;
until eoln;
      READLN;
write (file1,'U','S','I','N','G',' ','C','.');
writeln ('The field name____________');
repeat
      read (ch);
      if (ch <> '!') then begin
      write(file1)    end
      else goto 100;
until eoln;
      READLN;
write (file1,' ','=',' ','"');
writeln ('The field value____________');
repeat
      read (ch);
      if (ch <> '!') then begin
      write(file1)    end
      else goto 100;
until eoln;
      READLN;
write (file1,'"',';',' ');
write (filel,'E','N','D','_','S','T','O','R','E',' ');
writeln (' If you wish to make the transaction ');
write('parmanent type 1, otherwise type 0______');
read (m);
case m of
      1 : ho(c2);
      0 : ho(c3);
END;

reset (filel);
while ( not eof(filel)) do begin
read (filel, cl[i]);
i := i+1;     end;
i :=1;
ho(cl);
moveblank(cl);
goto 100;
END;                      (* of enter *)

'!'  : goto 100;
'8'  : show;

 otherwise
begin    writeln ('Please enter a valid option');
       goto 100;
end;
end;  (* of case *)
end;                      (* of edt *)

procedure process;
(* gives output of mathematical functions  *)

begin
writeln;
write('This facility allows you access to certain ');
write(statistical functions like max, min, count, total');
write('and average. It must be noted, however that these');
write('functions can take only numeric fields as
arguments');
write(' Besides sording is also provided both in the ');
write('descending as well as ascending order.');
writeln ('Press');
writeln ('    1_____find maximum');
writeln ('    2_____find minimum');
writeln ('    3_____count');
writeln ('    4_____find average');
writeln ('    5_____total');
writeln ('    6_____sort');
writeln ('    8_____show details of fields and relations);
writeln ('    !_____to opening menu');
writeln ('    CTRL/Y_____to system');
```

```
read(op);
case  op  of

'1':
begin
writeln ('MAX returns the largest of the values specified
');
write('by value expression for all the records specifiied');
write('by the RSE');
writeln ('Press RET key after entering any name. ');
invoke;
rewrite (filel);
ho(c3);
writeln (' Type the FIELD name________');
write(filel,'P','R','I','N','T',' ','M','A','X','
','M','.');
repeat
      read (ch);
      if (ch <> '!') then begin
      write(filel)    end
      else goto 100
until eoln;
      READLN;
WRITE (filel, 'O','F',' ','M',' ','I','N',' ');
WRITELN(' type the relation name______');
      repeat
      read (ch);
if (ch <> '!') then begin
      write(filel)    end
else goto 100;
        until eoln;
      READLN;
reset (filel);
while ( not eof(filel)) do begin
read (filel, cl[i]);
i := i+1;     end;
i :=1;
ho(cl);
moveblank(cl);
goto 100;
end; (* of max *)

'2': begin
writeln;
write('MIN returns the smallest of the values specified');
write('by the value expression for all the records ');
write('specifiied by the RSE");
write ('Press RET key after entering any name. ');
invoke;
rewrite (filel);
ho(c3);
```

```
writeln (' Type the FIELD name________');
write(filel,'P','R','I','N','T',' ','M','I','N','
','M','.');
      repeat
      read (ch);
if (ch <> '!') then begin
      write(filel)    end
else goto 100;
        until eoln;
      READLN;
WRITE (filel, 'O','F',' ','M',' ','I','N',' ');
WRITELN(' type the relation name_____');
      repeat
      read (ch);
if (ch <> '!') then begin
      write(filel)    end
else goto 100;
        until eoln;
      READLN;
reset (filel);
while ( not eof(filel)) do begin
read (filel, cl[i]);
i := i+1;     end;

i :=1;
ho(cl);
moveblank(cl);
goto 100;
end; (* of min *)

'3': begin
writeln ('COUNT returns the number of ');
write (' records in the stream specifiied by the RSE");
write ('Press RET key after entering any name. ');
rewrite (filel);
invoke;
ho(c3);
writeln (' Type the relation name_______');
write(filel,'P','R','I','N','T',' ','C','O','U','N','T','
');
WRITE (filel, 'O','F',' ',,'C',' ','I',N',' ');
      repeat
      read (ch);
if (ch <> '!') then begin
      write(filel)    end
else goto 100;
        until eoln;
      READLN;
reset (filel);
while ( not eof(filel)) do begin
read (filel, cl[i]);
```

```
i := i+1;     end;

i :=1;
ho(cl);
moveblank(cl);
goto 100;
end;

'4':
 begin
writeln('AVERAGE returns the average of the values ');
write('specified by the value expression for all the');
write('records specifiied by the RSE');
rewrite (filel);
write ('Press RET key after entering any name. ');
invoke;
ho(c3);
writeln (' Type the FIELD name________');
write(filel,'P','R','I','N','T',' ');
write(filel,'A','V','E','R','A','G','E');
write(filel,' ','A','.');
       repeat
       read (ch);
if (ch <> '!') then begin
       write(filel)    end
else goto 100;
         until eoln;
       READLN;
write (filel, 'O','F',' ','A',' ','I','N',' ');
writeln (' type the relation name_____');
       repeat
       read (ch);
if (ch <> '!') then begin
       write(filel)    end
else goto 100;
         until eoln;
       READLN;
reset (filel);
while ( not eof(filel)) do begin
read (filel, cl[i]);
i := i+1;     end;
i :=1;
ho(cl);
moveblank(cl);
goto 100;
end; (* of AVERAGE *)

'5':
 begin
writeln;
write('TOTAL returns the sum of the values specifified by');
write('value expression for all records specifiied by RSE');
rewrite (filel);
write ('Press RET key after entering any name. ');
invoke;
ho(c3);
writeln (' Type the FIELD name________');
write(filel,'P','R','I','N','T',' ','T','O','T','A','L');
write(filel,' ','T','.');
      repeat
      read (ch);
if (ch <> '!') then begin
      write(filel)    end
else goto 100;
        until eoln;
      READLN;
WRITE (filel, 'O','F',' ','T',' ','I','N',' ');
WRITELN(' type the relation name______');
      repeat
      read (ch);
if (ch <> '!') then begin
      write(filel)    end
else goto 100;
        until eoln;
      READLN;
reset (filel);
while ( not eof(filel)) do begin
read (filel, cl[i]);
i := i+1;     end;
i :=1;
ho(cl);
moveblank(cl);
goto 100;
end;          (* of TOTAL *)

'6':
 begin
writeln;
write('The SORTED BY cluase cuases RDO to arrange the ');
write('records in any order. A fieldname that determines ');
write('the sort order is called a sort key.');
write ('Press RET key after entering any name. ');
rewrite (filel);
invoke;
ho(c3);
rewrite(filel);
write (filel,'F','O','R',' ','S',' ','I','N',' ');
WRITELN ('The relation name____-');
repeat
      read (ch);
      if (ch <> '!') then begin
      write(filel)    end
```

```
          else goto 100;
            until eoln;
          READLN;
write (filel,'S','O','R','T','E','D',' ','B','Y',' ');
Writeln ('type either "ascending" or "descending"______');
          repeat
          read (ch);
if (ch <> '!') then begin
          write(filel)    end
else goto 100;
            until eoln;
          READLN;
write (filel,' ','S','.',' ');
writeln ('The SORT key___________');
          repeat
          read (ch);
if (ch <> '!') then begin
          write(filel)    end
else goto 100;
            until eoln;
          READLN;
write(filel,' ','P','R','I','N','T',' ','S','.');
writeln('Field name of which sorted values are required__');
          repeat
          read (ch);
if (ch <> '!') then begin
          write(filel)    end
else goto 100;
            until eoln;
          READLN;
write (filel,'E','N','D','_','F','O','R');
reset (filel);
while ( not eof(filel)) do begin
read (filel, cl[i]);
i := i+1;     end;
i :=1;
ho(cl);
moveblank(cl);
goto 100;
end; (* of SORTED *)

'!'  : goto 100;
'8'  : show;
 otherwise
begin    writeln ('Please enter a valid option');
        goto 100;
end;
end;  (* of case *)
end;          (* of process *)

(* of main program *)
```

```
begin

c2 := 'COMMIT';
C3  := 'ROLLBACK';
C4 := 'START_TRANSACTION READ_WRITE';

writeln ('                       MAIN MENU');
writeln ('Press the option number according to your
choice:');
writeln ('    1....To define the database');
writeln ('    2....To enter the data');
writeln ('    3....To edit');
writeln ('    4....To process data');
writeln ('    8....Show details of fields and relations);
writeln ('    CTRL/Y....To go back to the system');
writeln ('Now enter your option______');

read (op);
100 : begin
      mainmenu;
      end;
case op of
      '1' : format;
      '2' : store;
      '3' : edt;
      '4' : process;
        '5' : mainmenu;
      '8' : show;
 otherwise
begin    writeln ('Please enter a valid option');
       goto 100;
end;
end;  (* of case *)
end.
```

JAWAHARLAL NEHRU UNIVERSITY LIBRARY